NORTHERN BYTES

Volume 5 Number 3

Welcome to another issue of NORTHERN BYTES. Hopefully, this publication will be appearing on a somewhat more timely basis in the future. I expect that if all goes well, we will publish between four and six issues this year. There is a tradeoff involved - we can have more issues and fewer pages per issue, or fewer issues with more pages. I think that it is not unreasonable to expect this publication to appear on at least a quarterly basis, but more often than that might put a strain on the budget.

We want you to consider NORTHERN BYTES as your forum for exchanging ideas and public domain software. The TAS Public Domain Software Library is now a reality, and if you are interested in acquiring some good public domain software, send a self-addressed stamped envelope to TAS for information on the library. We are striving for quality rather than quantity in this library, so although we have relatively few public domain disks at present, those disks have many good programs on them.

Please feel free to contribute your public domain programs to the Library, and/or for publication in Northern Bytes. If you send us a program, please state whether it's intended for NORTHERN BYTES or for the Public Domain Library. If you don't specify, we'll assume we can use your program in either medium, or both. In fact, unless you tell us otherwise, we'll assume that any submission to NORTHERN BYTES may also be placed in the Public Domain Library.

We've got a lot of material to go into this issue, so I'm not going to waste any more space on long-winded opening remarks. I hope you enjoy this issue of NORTHERN BYTES!

NOTICE TO COMPUTER CLUBS AND USER GROUPS: Many computer clubs and user groups are receiving NORTHERN BYTES on an exchange basis. The word "exchange" implies just that - you send us your club or user group newsletter, we send you NORTHERN BYTES in return. Granted, we don't publish monthly, but generally speaking, an issue of NORTHERN BYTES usually contains at least two or three times as much material as the average computer club newsletter (you have my congratulations and respect if your club has an above-average newsletter!).

Some user groups have been getting a "free ride" - we've been sending NORTHERN BYTES, and those groups have not reciprocated by sending their newsletters. THIS WILL BE THE LAST ISSUE of NORTHERN BYTES that those groups will receive, unless we start receiving their newsletters. If, for some reason, your club is temporarily not publishing your newsletter (for example, if you're "between" editors), please write and explain the situation, and I will consider your case on an individual basis. If I don't hear anything from your group, though, this will be it!

Even if you think your group is exchanging newsletters with us, it might not hurt to have your club secretary check your mailing list. Due to the long delay between appearances of NORTHERN BYTES, we were dropped from a few exchange lists, and apparently have not been reinstated. I'd hate to drop a club from the mailing list due to a simple mixup.

I very much enjoy reading the newsletters from the various TRS-80 user groups, and hope to continue receiving one from your group!

THE EXTERMINATOR - This is where we kill the bugs that have been lurking from previous issues.

In Northern Bytes, Volume 5, Number 1, we ran an article entitled, "Color Computer and Model I/III Compatibility", which featured the NEWDOS/80 PDRIVE settings required to read Color Computer disks. Unfortunately, some of our readers took that to mean that they could pull a directory and read files from the CoCo disk. Sorry about that. When you use the PDRIVE settings mentioned, you can do only two things (or so I've been told): First, you can do a full disk copy from one CoCo disk to another, PROVIDED that BOTH disks are ALREADY FORMATTED!! You CANNOT format a CoCo disk, or copy anything less than the full disk. Second, I am told that you can use SUPERZAP to scan individual sectors on the CoCo disk. And that is ALL you can do, at least until one of you folks sends us some sort of neat utility program that will transfer files to and from a CoCo disk.

In our last issue of Northern Bytes (Volume 5, Number 2) we featured Tony Domigan's TRSDIR program (a program to pull a directory from a Model III TRSDOS 1.3 disk while using NEWDOS/80), which I had modified in order to make it run on the Model I and to add other improvements. Unfortunately, I outsmarted myself this time. Tony's original program used code similar to that shown below, which I replaced with a CALL 0FBDH instruction (this call converts a number stored in the arithmetic accumulator to a string for display purposes). I figured that I was saving a few bytes, since the code below and a CALL 0FBDH instruction produces the exact same format string with any positive integer. Since this code is used to display the number of free granules remaining on the TRSDOS disk, I figured it was safe to make the change because, after all, you can't have a negative number of free grans on a disk. Besides, the replacement routine worked every time I tested it.

Unfortunately, the routine at 0FBDH does not consider zero as a positive integer - technically, zero is neither positive or negative. It seems that the ROM routine at 0FBDH is a stickler for the technicalities, and the resulting string is in a slightly different format when zero is the value to be displayed. And, of course, it's quite possible to have zero grans of free space left on a disk. The program as published last month would display something like "0@#% Free Granules" in that situation. Naturally, in compliance with Murphy's Law, I didn't try to pull a directory off of a completely full TRSDOS disk before I published the program.

What to do to fix the bug? I took a look at Tony's original routine, put it back into the program, and found that it does work with a value of zero. He got it from Mumford Micro Systems' book, "Inside Level II", and they apparently got it from the ROM (starting at 0FB2H). Actually, I've made one change in the code - the ROM, "Inside Level II", and Tony Domigan all used an OR (HL) instruction. Since HL is set to 4130H in the subroutine at 1034H, I substituted an OR L instruction to reset the Z flag (it may not be shorter, but it saves seven whole T-states. After all, a true hacker has to be able to improve on the original code somehow!)

So - if you typed in the TRSDIR program from the last issue, please insert the following lines into the program to replace the CALL 0FBDH instruction, and re-assemble it. And, if you've tried use the ROM subroutine at 0FBDH and found that it didn't format the numeric string quite the way you wanted it to when zero was the value to be displayed, you may find that substituting the following code will do the trick for you. Here's the replacement code, which can be inserted into the program using the line numbers shown (line 1750 is replaced):

```
01750    XOR    A          ;Specify non-format edit
01752    CALL   1034H      ;Setup edit flag & buffer
01754    OR     L          ;Set NZ flag
01756    CALL   0FD9H      ;Change to numeric string
```

LETTER TO THE EDITOR - The following letter is from Greg Small, Box 607, Stouffville, Ontario, Canada L0H 1L0 . Greg's ideas should be of interest to you NEWDOS/80 users, so I am reprinting his letter here:

Dear Jack!

I was speaking to Charley last week about a Newdos/80 users group and he suggested I write to you with my thoughts.

+ Apparat appears to have ceased supporting Newdos/80.

+ There seems to be a large number of users of ND/80 and they have nowhere to turn for support or advice.

+ There is a need for support programs and support service for ND/80.

I propose the setting up of a loose knit international (world-wide) user group with approximately 25 regional bulletin

boards as the base. Each board would further service local boards.

These regional boards would collect and disseminate information as available. If a question was asked or information became available locally that was not available to the group as a whole the local collector would pass this on to either a central clearing house or to the next board in the chain, who would then pass it on to the next, until the message had made it back to its starting point.

The local boards would access the regional boards for information. As well, there would be a "published" newsletter that would further serve to collect all information as hardcopy, to be available to those without communication facilities on their computer.

I envisage a nominal membership fee to defray the costs of the regional or local boards information exchanges. This would be something on the order of $10 per year with a small percent going to the international organization to defray the cost of keeping the whole thing together.

I would like to see it set up as a non-profit organization with perhaps some corporate sponsorship in exchange for advertising in the newsletters. In the event that the organization made substantial money the costs would be reduced for the following year.

We appear to, collectively, have a vast amount of information. But, this is not available to the Newdos/80 community as a whole due to the lack of an organization. It also seems that many of us duplicate the work of others.

I also feel that large numbers of users will not upgrade their system for the forseeable future as they have everything they want in their present system. Thus, it is imperative that Newdos/80 be kept alive.

Please let me have your thoughts on this. I have sent Charley a copy as I wanted him to be kept up to date on this.

Best regards, Greg Small

(Voice-7pm-9pm 416-640-4700, Data-24 hours 416-640-3434)

Editor's comments: I feel that Greg's ideas have merit for two reasons. One is that considering the large number of NEWDOS/80 users, it is very surprising that you seldom see NEWDOS/80 user groups on either a local or national level. One factor in this may be that NEWDOS/80 is easy to use, something that cannot be said for some of the other disk operating systems (perhaps the users of some of those operating systems HAVE to band together in order to figure out how to operate their DOS). Nevertheless, I feel that a NEWDOS/80 user organization would be a worthwhile source of information.

Also, Greg proposes the "networking" together of many BBS systems - something that should be a reality already in the BBS community, but in fact is done rarely or not at all. I have often wondered why a BBS user cannot leave a message on his local BBS for a user of another (distant) BBS, and have the message forwarded (over a period of time) to the receipient's local BBS system. A message base could be "shared" among more than one BBS. As far as I know, there is no BBS software available that really embraces the concept of sharing a message base among more than one system. The actual transfer of messages could be either on a general basis (all BBS's eventually receive all messages) or on a "directed" basis (messages could be "addressed" to a specific receipient BBS), or a combination of both, and the actual transfer of messages could be either by telephone (perhaps at night when rates are low) or by mailing disks containing the message base files. If a number of BBS's could be networked together, a "master" BBS could use outward WATS lines (or MCI, SPRINT, etc. service) to call all "satellite" systems once or twice each night (perhaps once to receive all incoming messages, then after "sorting" the messages it would call back each BBS and "leave" the messages directed to that BBS). Those of you into telecommunications might enjoy the challenge of writing the software to accomplish something like this.

But, to get back to the original thought of Greg's letter, I hope that those of you that use NEWDOS/80 and that feel you could benefit from belonging to a nationwide users group will contact Greg and register your support for this project. If you don't, then don't complain the next time you need some assistance and have nowhere to turn!

MOVING MODEL 4 ROM INTO RAM - The hardware design of the Model 4 makes it possible to "flip" the ROM code into the Model 4's low RAM, so that Model III BASIC becomes RAM-based, which in turn makes it possible to patch BASIC or other parts of the ROM (such as the device drivers). I use this capability in my VIDEO4 program, in order to patch certain BASIC commands and functions to operate correctly in a 24x80 display environment.

This works fine as long as you don't use LDOS 5.1 or NEWDOS/80 as your Disk Operating System. However, both of the aforementioned DOSes use the ROM area (starting at 0000H or 0100H, respectively) as a "bit bucket" during disk write-with-verify operations, so when you have moved the ROM code into RAM, LDOS or NEWDOS/80 will bomb it as soon as you do any disk write operation (including a write to the disk directory, as would happen when a file is KILLed or RENAMEd). The following patches move the location of the "bit bucket" from 0000H or 0100H to 3800H (the memory-mapped keyboard area, which is still considered a "read-only" section of memory). I'm not certain, but I suppose it's remotely possible that under certain conditions, LDOS or NEWDOS/80 might use a "bit bucket" of more than the 1K of keyboard memory located between 3800H and 3BFFH. If that were to happen, garbage would be written into the video display memory (starting just above the keyboard memory, at 3C00H), so you would have a visual indication of the problem. If you actually experience this phenomenon, I'd like to hear from you. Here are the patches:

LDOS 5.1 for the Model III: Type the following line from LDOS READY:

PATCH SYS0/SYS.RS0LT0FF (X'46C2'=38)

(NOTE: You may wish to verify that this byte is the correct location to change before applying this patch. If so, use DEBUG to examine the three bytes starting at 46C1H in memory. Prior to the application of this patch, they should be 26 00 CD. Also, note that the password RS0LT0FF used in the above command line contains two ZEROES, not two letter "O"s).

NEWDOS/80 version 2.0 for the Model III: Use the DFS function of SUPERZAP to make the following zap to SYS0/SYS, file sector 02, byte AB:

change: 26 01 CD to: 26 38 CD

(NOTE TO MODEL I USERS: This same zap can be applied to the Model I version of NEWDOS/80, but the location to patch is SYS0/SYS, file sector 03, byte 2A, so if you are a Model I owner with a hardware modification that allows you to move ROM into RAM, you also can fix NEWDOS/80 to work with your system.)

PLEASE NOTE that these patches have NOT been completely tested. If you apply them, please test them thoroughly before you use them with any irreplaceable programs or data.

Thanks to GREG SMALL from Stouffville, Ontario, Canada for his assistance in helping to discover the source of the problem. NO thanks to Apparat, because they offered absolutely no assistance in tracking down the problem (it seems that they are now pursuing the IBM market, and don't have much interest in providing further support for the TRS-80 products that made them rich in the first place).

ANOTHER VIDEO4 PATCH - If you prefer TRSDOS 1.3 to NEWDOS/80 (see above item), try this patch to fix up the directory display format when using VIDEO4. It does not seem to affect the normal directory display when VIDEO4 is not in use. From TRSDOS READY, enter the following command:

PATCH *6 (ADD=5AFA,FIND=3F,CHG=0F)

VIDEO4 UPGRADE AVAILABLE FREE! - If you are a VIDEO4 owner, and you did NOT receive a registration form when you purchased VIDEO4 (or if your copy of VIDEO4 does not display a copyright notice and version number when you first run it), you may return your master VIDEO4 disk and a self-addressed, stamped disk mailer (or whatever you use to mail disks) to TAS for a FREE upgrade. You MUST enclose the stamped, self-addressed disk mailer or your disk will NOT be returned (you have been warned!). There is no other charge for the upgrade. If you live outside the U.S.A., you may send an unstamped mailer and 54 cents in U.S. coins (or 65 cents in Canadian coins) - if you live outside the U.S.A. or Canada, that will buy surface delivery only (small packet rate).

The upgraded version of VIDEO4 supports the on-screen clock display (available from DOS through use of a CLOCK ON or CLOCK Y command), and also makes the clock keep the correct time when the fast clock speed is used. It also causes the computer to emit an audible "beep" when certain BASIC errors occur. Along with the upgrade, you will receive a registration form which will allow you to register your copy of VIDEO4. You'll

also receive a PATCH UPDATE sheet which will inform you of the one-byte patch you need to make to your DOS in order to use VIDEO4, if you use NEWDOS/80, LDOS, or TRSDOS 1.3 (these patches are the same ones mentioned in previous paragraphs of this newsletter).

Let me also mention a couple of other DOS compatibility notes - if you use VIDEO4 with TRSDOS 1.3, you should probably NOT use the fast clock speed option, unless you like saving garbage instead of good data to your disk files. And, if you use VIDEO4 with MULTIDOS, make sure you have the current version of MULTIDOS. If you do not have MULTIDOS version 1.6b (or later), send your master disk and a $5.00 upgrade fee to Cosmopolitan Electronics (NOT to TAS) to get upgraded to the current version.

Some of you may be wondering, "what is VIDEO4?" The answer is that it's a program that permits use of the 24X80 video display mode and fast clock speed of the Model 4, while in the Model III mode. Although there are similar packages available that are less expensive, VIDEO4 is a far superior package, and COMMENTED EDITOR-ASSEMBLER SOURCE CODE is included on the disk, in case you want to "tweak" the program a bit! Contact The Alternate Source for further information on the VIDEO4 package.

ADDING YOUR OWN /SYS OVERLAYS TO NEWDOS/80 by Jack Decker - The purpose of this article is to give you some basic information you may have wished you'd had if you ever wanted to add your own /SYS overlays to a Disk Operating System. This article is written for those that already have some familiarity with DOS operations - specifically, I will assume that you know how to use the SUPERZAP program supplied with NEWDOS/80.

You might wonder why you'd want to add a /SYS overlay to your DOS. The answer is that you might want to add a command or function to your operating system that requires more code than that which can be placed in a simple patch to an existing system file. Actually, I had a somewhat similar problem, which prompted me to ferret out this information. As you may be aware, TRS-80 Disk BASIC uses an "ampersand (&) function" to convert hexadecimal or octal constants to variables. Its application is fairly limited, for example, you can't use the INPUT statement to input a string of hexadecimal digits and then use the &H function to convert them to decimal, because the &H function only works with constants placed within the program itself, not with variables. So, I wrote an improved ampersand function routine, which appears in Appendix VI of my book, "TRS-80 ROM Routines Documented." The only problem with the new routine was that it had to be loaded into high memory after calling up BASIC from DOS READY. Since I usually didn't bother to do this, the routine was never available when I wanted to use it. What I wanted to do was to make my "improved" ampersand routine a more-or-less permanent part of Disk BASIC.

Even though BASIC is, strictly speaking, an application program (that is, a visible /CMD file) rather than a part of the DOS, there are nevertheless several /SYS overlays that are on the disk solely to support and provide additional functions for BASIC. Since BASIC can call and use those /SYS overlays, I figured that I ought to be able to convert my ampersand function routine into a /SYS overlay that BASIC could access at any time.

The following paragraphs describe how I did this with NEWDOS/80. If you are using another DOS, the actual method may be different (and you are on your own), but some of the principles involved may be similar. The NEWDOS/80 manual has a discussion of user coded system routines in section 12.6.1, but it is so brief that it is almost unintelligible (as a matter of fact, I didn't even know it was there until someone called my attention to it after reading a first draft of this article - which was probably a good thing, because I think I would have been more confused had I read the manual discussion first).

First of all, I made a few changes to the source code of the ampersand function routine as published in the book (if you don't have the book, you're in luck - sort of. The routine had a bug in it so the corrected version is being printed elsewhere in this newsletter. However, if you need the instructions for using the routine, you'll have to get your hands on a copy of my book). In addition to fixing the bugs in the routine, I deleted lines 100-140, changed the ORG address in line 170 to 4D00H, and changed line 1260 (now line 1250 in the corrected version) from END 06CCH to END START. Then I assembled the program using the filename SYS29/SYS:0 . At this point I had created a usable DOS overlay with the name SYS29/SYS, which resided on the system disk in drive zero, and which when loaded would reside in the DOS overlay area starting at 4D00H. The only problem was, BASIC had no idea that it was there, or that it was now supposed to use this overlay to interpret any "&" function call.

Time to patch the DOS exit for the "&" function. This is found at 4194H, and in the Model I version of NEWDOS/80 normally contains a JP 5790H instruction. Instead of going there, we want it to call our new SYS29/SYS overlay and execute it, so we simply change this to a LD A,3FH instruction followed by a RST 28H instruction. RST 28H is used for DOS overlay requests (among other things), on entry, the value in the A register indicates the DOS overlay to be called. Wait a minute - how do we get 3FH out of system overlay number 29 (SYS29/SYS), I hear you ask. Well, for one thing, the actual filename used is not important here - once we get this working, we could change the filename SYS29/SYS to TOADSTL/FRG or something equally ridiculous and it would still work - I'll explain later. But, for the time being, consider the byte in the A register as TWO numbers - one in the most significant three bits, and the other (the /SYS overlay number) in the least significant five bits. In other words, our eight-bit byte looks like this:

UUUSSSSS

The bits in UUU are a "user code", and are defined as such so that one overlay can be used by more than one command or function, as will be explained later. These three bits can be set to any value EXCEPT 000 (zero), meaning that user codes from one through seven (decimal) are permissible. Since we are only using this overlay for one function, we can use any non-zero value we want to, so for the sake of argument, let's set the user code to "1" (that's 001 binary).

Bits SSSSS are the system overlay number PLUS TWO. So, in this case, we have a "user code" of 001 binary and an overlay number of 29 + 2 = 31 decimal = 1F hexadecimal or 11111 binary. Put those binary numbers together and we have 001 11111 binary, or 3F hexadecimal - which is why we load the A register with 3FH before doing the RST 28H. Note that the LD instruction is two bytes long and the RST instruction is only one byte long, so this code fits nicely into the DOS exit for the "&" function.

We could, of course, POKE the proper code into the DOS exit each time we enter BASIC, but we can make the change permanent by using SUPERZAP to patch BASIC/CMD. In the Model I, the patch is as follows:

BASIC/CMD,17,44 change 64 C3 90 57 C3 to 64 3E 3F EF C3

Model III users may need to apply the patch in a different location, if so, the F (find) function of SUPERZAP may aid you in finding the proper location to patch.

Hold it, Fillmore, we're not quite through yet. If you go into BASIC and try to use the "&" function, you'll probably get a SYSTEM PROGRAM NOT FOUND message and then you'll get thrown back to DOS READY. The reason is that the DOS does not look for its /SYS overlays by using the filename. Instead, it expects to find SYSn/SYS in directory slot number n+2 (that's why we had to add two to the system overlay number in the above paragraph). If the first byte of that directory slot is 5FH, indicating that the protection level of the program is "system", then the DOS could care less what the actual filename is - it figures that that's the SYS overlay it wants, and goes ahead and uses it. So, before we can use our new SYS29/SYS, we have to get its directory entry into the proper slot, and we have to protect it as a /SYS file.

Time to call up SUPERZAP again. Use SUPERZAP'S DFS function to examine DIR/SYS:0, and use the + key to page through the directory. Find SYS0/SYS, then SYS1/SYS, then SYS2/SYS, and so on. You'll soon notice a pattern - with the exception of the first two sectors of the directory (the GAT and HIT sectors), the /SYS files are arranged in ascending order through the DIR/SYS file. It's easier for you to discover the pattern by examining DIR/SYS with SUPERZAP than it is for me to try and explain it, so go ahead - I'll wait.

Oh, you're back. Well, if you're any kind of a programmer at all, you've probably already figured out which directory slot your SYS29/SYS file has to occupy - it's the slot just below SYS21/SYS on the SUPERZAP display (from the top of the display you'll see SYS5/SYS, SYS13/SYS, SYS21/SYS, and the slot just below that is where SYS29/SYS must go). So, how do you get SYS29/SYS into that directory slot? Well, you could copy the SYS29/SYS directory entry from its present location into that slot, then zero the original slot and use SUPER-UTILITY or a similar program to fix the Hash Index Table. You could reread Pennington's book and

rebuild the directory to proper specifications. Or, you could cheat a bit and do it the lazy man's way. I took the latter route.

Basically, it works like this. First of all, let's assume that there's already a valid directory file entry occupying the slot that SYS29/SYS should be in. Copy that file to another disk (you can copy it back to your system disk later). If there is no valid directory file entry in that slot (no entry at all, or the entry of a previously KILLed file, as indicated by the first byte of the entry containing a value of zero), then use the CREATE command and keep CREATING files (create five or ten at a time - use short filenames like Q1, Q2, etc.) until you get a valid file to occupy that directory slot. Check with SUPERZAP to make sure that the first byte of that directory slot is a non-zero value.

Once you have the directory entry of a valid file (one that has not previously been KILLed) occupying the directory slot where SYS29/SYS should be, note the filename presently in that slot (let's suppose it's OLDFILE/CMD). Be sure to copy OLDFILE/CMD to another disk if you want to save it, then issue a command of the form: COPY SYS29/SYS TO OLDFILE/CMD . Then KILL SYS29/SYS and then RENAME OLDFILE/CMD TO SYS29/SYS . You should now have SYS29/SYS in its proper slot (use SUPERZAP to check), but it still isn't protected as a system file. To do that, use SUPERZAP to change the first byte of the directory entry from whatever it is now (probably 10) to 5F. You may also wish to drop down to the second line of the directory entry and change the password bytes (the first four bytes on the second line) so that they are the same as those in the directory entries for the other /SYS files on that disk.

Don't forget copy the file you moved to another disk (if any) back to your system disk, and/or to kill all of those "garbage" files you CREATEd in a previous step (if any).

Admittedly, this process is a lot easier to do than to describe, particularly if you're already familiar with SUPERZAP and with the organization of the disk directory. The whole point of the above few paragraphs was to get our new SYS29/SYS file into the proper directory slot where the DOS could find it. If you have an easier method of accomplishing this, by all means use it (then write it up and send it in so the rest of us can use it).

I mentioned that an overlay could be used for more than one function. Suppose that we combined the ampersand routine mentioned above with a two-byte POKE command (which would complement the &PEEK function of the ampersand routine). We'll use the command NAME POKE because NAME is a reserved BASIC keyword that is unused in NEWDOS/80, and is vectored out to reserved RAM (in much the same way that the ampersand function is, except that NAME is considered a BASIC command, while the ampersand function is just that - a function. This is an important distinction, because PEEK is a function while POKE is a command. In BASIC, you must always "do something" with the result of a function - PRINT it, assign it to a variable, etc. - but the same is not true of a command).

Let's use the same overlay for both the ampersand function routine and the NAME POKE routine - the only difference is that we'll continue to use a user code of one for the ampersand function routine, but we'll assign a user code of two (actually any valid user code other than one) to the NAME POKE routine. Both routines will be part of SYS29/SYS, but when we want to call the NAME POKE routine, the user code two (010 binary) will be combined with the bits in SSSSS (still 11111), giving us a combined code of 010 11111 binary or 5F hexadecimal. So, when we want to use our NAME POKE function, we would simply load the A register with 5FH and do a RST 28H. This would normally be done from the NAME command DOS exit at 418EH - if you're patching BASIC, the Model I patch would be:

BASIC/CMD,17,3E change 57 C3 4A 1E C3 to 57 3E 5F EF C3

The contents of the A register are not destroyed by the act of loading the DOS overlay, so at the very start of the overlay we can test the value of A and jump to the appropriate section of the overlay. This is done in the following section of code. To add this code to the ampersand function routine, first delete all lines up to and including line 180 in the present version of the routine, then add the code shown below to the start of the routine. Then assemble it as SYS29/SYS using the same method detailed above:

```
00060          ORG   4D00H       ;System overlay area
00070 START    CP    3FH         ;Called from & function?
00080          JR    Z,AMPRFN    ;Go if so
00090          RST   8           ;Syntax Error if "POKE"
00100          DEFB  0B1H        ;  doesn't follow "NAME"
00110          CALL  2B02H       ;Get address to POKE
00120          PUSH  DE          ;Save address to POKE
00130          RST   8           ;Syntax Error if comma
00140          DEFB  ','         ;  doesn't follow address
00150          CALL  2B02H       ;Get value to POKE
00160          EX    (SP),HL     ;Save in ptr,get POKE adr
00170          JP    2860H       ;Finish up in ROM
00180 AMPRFN   RST   10H         ;Get next character
```

Note that at the start of this section of code, we tested the A register for 3FH (a user code of one, indicating that the ampersand function was called) and if we did not find it, we assumed that the A register contained 5FH (user code two) since those are the only two ways that this overlay can be entered. However, note that values of 3FH, 5FH, 7FH, 9FH, BFH, DFH, and FFH could all be used to access overlay SYS29/SYS, and that those values correspond to the seven possible user codes. At the start of the overlay, we could test for the various possible values and jump to the appropriate routine. So, one overlay could hold up to seven different routines, which need not be related to each other in any way other than being contained within the same overlay file!

The point of all this is that sometimes when you want to add an extension to the DOS or to Disk BASIC, it's not always necessary to make it high-memory resident - particularly if it's code that's only used occasionally for a specific purpose. Think about using overlays for those "once in a while" applications!

"TRS-80 ROM ROUTINES DOCUMENTED" BUG - It's been said that no program is ever fully debugged, and I'm beginning to believe it. For example, I wrote an "Improved Ampersand Function" routine that ultimately became Appendix VI of my book. I tested it every way I could think of, and it seemed to work just fine. Others have used it for months, and it was reprinted in a computer club newsletter, and I hadn't heard of anyone having problems with it. That's why I was a bit incredulous when Nathan Harrington phoned me from Lincoln, Nebraska to tell me that he had found a bug in it. As it turned out, he was right.

It seems that if the routine is enabled and the following line is executed from BASIC, the system bombs with a "STRING FORMULA TOO COMPLEX ERROR" on the tenth loop:

```
10 FOR A=1 TO 20: PRINT &H(&FN(A)): NEXT
```

Nathan thought that the &FN routine (which converts a number to a hexadecimal string) was the source of the problem - a logical assumption, but not correct. Actually, the &H(string) routine was the culprit. Nathan also thought that the string workspace in reserved RAM was not being cleaned up properly, and that DID turn out to be correct. In any event, I sat down with my copy of "Model III ROM Commented" (which I could probably sell a thousand copies of if I had them to sell, since the publisher, Soft Sector Marketing, is no longer in business), and discovered a handy ROM routine (what else?) that not only did the job of cleaning up the workspace and eliminating the bug, but also did some of the work that I had been doing in my routine. This actually allowed me to shorten the routine by a few bytes.

The listing of the revised program is below. Changes from the book version are as follows: The ORG address in line 170, and the changed code in lines 770-810. Also, line 820 and all lines following were numbered greater by 10 in the book version (820 was 830, etc.). And last (and probably least), the glaring error in the comment in line 100 has been fixed (it's not a link to the USR function vector...).

The text in the book is still correct, with the exception of the MEMORY SIZE of 32591 in the last paragraph, which now reserves a bit (well, four bytes, actually) too much memory!

The revised program listing follows. I am sorry for any inconvenience this may have caused anyone. Please advise me if you have any further problems with this routine.

```
            00100 ;       LINK TO & FUNCTION VECTOR
            00110
4194        00120         ORG   4194H          ;VECTOR FROM "&" FUNCTION
4194 C3537F 00130         JP    START          ;JUMP TO START OF ROUTIN
            00140
            00150 ;       MAIN PROGRAM BEGINS HERE
            00160
7F53        00170         ORG   7F53H          ;MAIN PRGM-MAY RELOCATE
7F53 D7     00180 START   RST   10H            ;GET NEXT CHARACTER
```

```
7F54 CB7F   00190          BIT   7,A          ;CHECK FOR TOKEN
7F56 2844   00200          JR    Z,NOTTKN     ;GO IF NOT BASIC TOKEN
7F58 F5     00210          PUSH  AF           ;SAVE TOKEN
7F59 D7     00220          RST   10H          ;GET NEXT CHARACTER
7F5A CD2C25 00230          CALL  252CH        ;EVALUATE EXPRESSION
7F5D E3     00240          EX    (SP),HL      ;SAVE BASIC POINTER
7F5E E5     00250          PUSH  HL           ;RE-SAVE TOKEN
7F5F CD7F0A 00260          CALL  0A7FH        ;CHANGE TO INTEGER
7F62 F1     00270          POP   AF           ;GET TOKEN
7F63 FEE5   00280          CP    0E5H         ;2-BYTE PEEK?
7F65 2009   00290          JR    NZ,NOTPK     ;IF NOT 2-BYTE PEEK
7F67 5E     00300          LD    E,(HL)       ;GET CONTENTS OF ADDRESS
7F68 23     00310          INC   HL           ;  POINTED TO BY HL AND
7F69 56     00320          LD    D,(HL)       ;  PUT IN DE REGISTERS
7F6A ED532141 00330        LD    (4121H),DE   ;  AND MATH BUFFER
7F6E E1     00340          POP   HL           ;RESTORE BASIC POINTER
7F6F C9     00350          RET                ;NUMBER IN BUFFER & DE
7F70 FEBE   00360 NOTPK    CP    0BEH         ;CHECK FOR HEX CONVERSION
7F72 C29719 00370          JP    NZ,1997H     ;SN ERROR IF NOT HEX CONV
7F75 EB     00380          EX    DE,HL        ;PUT VALUE IN DE
7F76 CD807F 00390          CALL  CONV         ;CONVERT TO STRING
7F79 CD9310 00400          CALL  1093H        ;MARK END OF STRING
7F7C C5     00410          PUSH  BC           ;FOR ROM TO DISCARD
7F7D C33928 00420          JP    2839H        ;RETURN THRU STR$ ROUTINE
7F80 213041 00430 CONV     LD    HL,4130H     ;ASCII CONV. WORKSPACE
7F83 7A     00440          LD    A,D          ;CONVERT D REGISTER
7F84 CD887F 00450          CALL  CONV2        ;  TO ASCII (HEX)
7F87 7B     00460          LD    A,E          ;CONVERT E REGISTER
7F88 F5     00470 CONV2    PUSH  AF           ;SAVE BYTE TO CONVERT
7F89 0F     00480          RRCA               ;ROTATE HIGH NYBBLE
7F8A 0F     00490          RRCA               ;  DOWN TO LOWER
7F8B 0F     00500          RRCA               ;  FOUR BITS
7F8C 0F     00510          RRCA
7F8D CD917F 00520          CALL  CONV3        ;CONVERT TO ASCII (HEX)
7F90 F1     00530          POP   AF           ;RESTORE BYTE TO CONV
7F91 E60F   00540 CONV3    AND   0FH          ;USE LOWER FOUR BITS ONLY
7F93 C690   00550          ADD   A,90H        ;THIS ROUTINE CONVERTS
7F95 27     00560          DAA                ;  HEX VALUE IN RANGE
7F96 CE40   00570          ADC   A,40H        ;  00H-0FH TO ASCII CHAR
7F98 27     00580          DAA                ;  0-9 OR A-F
7F99 77     00590          LD    (HL),A       ;STORE RESULT ASCII CHAR
7F9A 23     00600          INC   HL           ;BUMP WORKSPACE POINTER
7F9B C9     00610          RET                ;(THIS ROUTINE RECURSIVE)
7F9C 010201 00620 NOTTKN   LD    BC,102H      ;SET UP BINARY PARAMETERS
7F9F FE42   00630          CP    'B'          ;IS IT BINARY?
7FA1 280F   00640          JR    Z,CONT       ;GO IF BINARY
7FA3 011004 00650          LD    BC,410H      ;SET UP HEX PARAMETERS
7FA6 FE48   00660          CP    'H'          ;IS IT HEXADECIMAL?
7FA8 2808   00670          JR    Z,CONT       ;GO IF HEXADECIMAL
7FAA 010803 00680          LD    BC,308H      ;SET UP OCTAL PARAMETERS
7FAD FE4F   00690          CP    'O'          ;IS IT OCTAL?
7FAF 2801   00700          JR    Z,CONT       ;GO IF OCTAL
7FB1 2B     00710          DEC   HL           ;OCTAL ASSUMED
7FB2 D7     00720 CONT     RST   10H          ;GET NEXT CHARACTER
7FB3 FE28   00730          CP    '('          ;IS CHARACTER A "("?
7FB5 2018   00740          JR    NZ,NOTEXP    ;GO IF NOT A "("
7FB7 C5     00750          PUSH  BC           ;SAVE COUNTER & MAX DIGIT
7FB8 CD2C25 00760          CALL  252CH        ;EVALUATE EXPRESSION
7FBB E3     00770          EX    (SP),HL      ;SAVE BASIC BYTE POINTER
7FBC E5     00780          PUSH  HL           ;RE-SAVE CNTR & MAX DIGIT
7FBD CD072A 00790          CALL  2A07H        ;CLEAN UP WORKSPACE
7FC0 C1     00800          POP   BC           ;RESTORE CNTR & MAX DIGIT
7FC1 5F     00810          LD    E,A          ;GET STRING LENGTH IN E
7FC2 23     00820          INC   HL           ;GET ADDRESS OF FIRST
7FC3 7E     00830          LD    A,(HL)       ;  CHARACTER OF STRING
7FC4 23     00840          INC   HL           ;  AND STORE IN HL
7FC5 66     00850          LD    H,(HL)       ;  REGISTERS
7FC6 6F     00860          LD    L,A
7FC7 EB     00870          EX    DE,HL        ;HL=STRING LNGTH, DE=STRT
7FC8 19     00880          ADD   HL,DE        ;HL=STRING END + 1
7FC9 EB     00890          EX    DE,HL        ;HL=START, DE=END + 1
7FCA CDD27F 00900          CALL  CONT2        ;EVAL. STRING EXPRESSION
7FCD E1     00910          POP   HL           ;RESTORE BASIC POINTER
7FCE C9     00920          RET                ;NUMBER IN MATH BUFFER
7FCF 11FFFF 00930 NOTEXP   LD    DE,0FFFFH    ;CHECK TO NON-HEX CHAR.
7FD2 2B     00940 CONT2    DEC   HL           ;BACK UP BASIC POINTER
7FD3 E5     00950          PUSH  HL           ;SAVE BASIC POINTER
7FD4 210000 00960          LD    HL,0         ;ZERO MATH ACCUMULATOR
7FD7 CD9A0A 00970          CALL  0A9AH        ;  & SET TYPE FLG TO INT.
7FDA E3     00980          EX    (SP),HL      ;HL=BASIC POINTER, (SP)=0
7FDB DDE1   00990 NXTDGT   POP   IX           ;STORE # SO FAR IN IX
7FDD D7     01000          RST   10H          ;GET NEXT CHAR. (DIGIT)
7FDE 08     01010          EX    AF,AF'       ;SAVE CHARACTER & FLAGS
7FDF DF     01020          RST   18H          ;CHECK FOR END OF STRING
7FE0 D0     01030          RET   NC           ;RETURN IF END OF STRING
7FE1 08     01040          EX    AF,AF'       ;RESTORE CHAR. & FLAGS
7FE2 3805   01050          JR    C,DIGIT      ;IF CHAR. IN RANGE 0 TO 9
7FE4 FE41   01060          CP    41H          ;IS CHAR BELOW ASCII "A"?
7FE6 D8     01070          RET   C            ;END OF NUMBER IF < "A"
7FE7 D607   01080          SUB   7            ;OFFSET FOR ALPHA CHARS.
7FE9 D630   01090 DIGIT    SUB   30H          ;A=0 TO 15 FOR 0-9 OR A-F
7FEB B9     01100          CP    C            ;C=2, 8, OR 10H MAX DIGIT
7FEC D0     01110          RET   NC           ;END OF NUMBER IF >= MAX.
7FED DDE5   01120          PUSH  IX           ;PUT # SO FAR ON STACK
7FEF E3     01130          EX    (SP),HL      ;HL=# SO FAR, (SP)=PNTR
7FF0 C5     01140          PUSH  BC           ;SAVE COUNTER & MAX DIGIT
7FF1 29     01150 TIMES2   ADD   HL,HL        ;MULTIPLY HL TIMES 2
7FF2 DABZ07 01160          JP    C,7B2H       ;ERROR IF OVERFLOW >FFFFH
7FF5 10FA   01170          DJNZ  TIMES2       ;REPEAT MULTIPLY 'TIL B=0
7FF7 4F     01180          LD    C,A          ;ADD VALUE OF LATEST
7FF8 09     01190          ADD   HL,BC        ;  DIGIT FETCHED TO HL
7FF9 222141 01200          LD    (4121H),HL   ;CURRENT HL TO BUFFER
7FFC C1     01210          POP   BC           ;RESTORE CNTR & MAX DIGIT
7FFD E3     01220          EX    (SP),HL      ;HL=BASIC PNTR, (SP)=#
7FFE 18DB   01230          JR    NXTDGT       ;GET NEXT DIGIT (IF ANY)
            01240
06CC        01250          END   06CCH        ;USE 1A19H FOR MODEL III
            01260                             ;  OR 402DH FOR DOS

00000 TOTAL ERRORS

CONT   7FB2   CONT2  7FD2   CONV   7F80   CONV2  7F88
CONV3  7F91   DIGIT  7FE9   NOTEXP 7FCF   NOTPK  7F70
NOTTKN 7F9C   NXTDGT 7FDB   START  7F53   TIMES2 7FF1
```

TRS-80 ROM ROUTINES DOCUMENTED UPGRADE - If you have purchased my book and your copy does not have the "Hexadecimal Address Cross-Reference" on pages 122-125, you have a copy of the first printing. You may receive an upgrade free of charge by removing the last page from your book (the one with the "Afterword" on it - this page is replaced in the upgrade) and sending it AND a large (approximately 9" by 12"), self-addressed stamped envelope with 54 cents postage to me at TAS. Alternately, if you don't have the large envelope send 65 cents U.S. (or 80 cents Canadian) in coin and a pre-addressed mailing label and I'll supply the envelope and stamps. Please be patient, I'll probably wait until I've received several requests and then send them all out at once.

Also, it has come to my attention that a few copies of my book that were of substandard quality were inadvertently shipped out. If you got a copy that has pages that are difficult to read (particularly if they are extra light at the top or bottom of the page), please feel free to return it for a replacement copy at no charge (we'll even pay the postage). On the other hand, if your copy has only one or two bad pages, you may elect to tell us which pages are bad and we'll send you replacements. If you do want to return your present copy for replacement, you may want to wait a couple of months just in case I discover any more bugs and make further revisions, but that's entirely up to you. However, I do apologize to anyone that got a bad copy (by the way, if you ordered my book and had to wait a while to get it, it was because we had some real quality-control problems for a while. We have had to scrap practically two entire print runs due to these problems! So, my apologies to anyone that had to wait - and a double apology if you had to wait and still got a bad copy!).

NEWDOS/80 TIP by R. Barto is excerpted from NYBBLER - If you use Newdos/80 and use the command "Format [dn]" you will notice that the disk will have a name of "NOTNAMED" and a date of "00/00/00" [only if you did not set the system date when you booted the DOS -ed.]. You can change both of these with the command "PROT [dn] Name=Newname Date=newdate", but if you want to have a new default name, you can with the added feature of lowercase and punctuation. Using SUPERZAP, enter DFS, then SYS6/SYS, then MOD 3D. You can change all bytes to byte 44 (8 bytes) to whatever you want. Do not change the "B" at 3C, only 3D-44 replacing the word "NOTNAMED". The entries must be in hexadecimal, and you can use the trial and error method if needed. I use "Bob'sfmt" and love it.

A 6809 ASSEMBLER IN MICROSOFT BASIC

by Clarence Felong and Ken Brown

When the Radio Shack Color Computer was first announced, it appeared to be a break-through in price/performance. It was based on the Motorola 6809 CPU, had color graphics, Basic in ROM and cost under $500! At about this same time we became very interested in graphics and animation (i.e. games). The Basic was good, faster then the TRS-80's, but not fast enough to handle real-time animation. Assembly language programming seemed the way to go.

BACKGROUND

A problem arose very quickly. Where do you get a 6809 assembler to run on the Color Computer when the Advanced Basic and add-on RAM was not even available? What could the 6809 DO, anyway? After many phone calls to Radio Shack, Motorola part distributors and local book stores, we obtained a book entitled, "MC6809 PRELIMINARY PROGRAMMING MANUAL" from Motorola. Preliminary, indeed! The listings in the book came from a 6809 cross-assembler running on a 6800 based machine and many of the comments were hand written. This was interesting but did not solve our problem.

Our pleas to Radio Shack for information on availability of an assembler went unanswered. South West Technical Products had a 6809 based computer running on the SS-50 bus but we were unwilling to spend several thousand dollars for a computer to write code for another computer. Spectral Associates had advertised an assembler but were "weeks" away from delivery at that time. It appeared that our options were to wait or write our own. We decided to write our own.

The Color Computer did not then have the memory capacity to run a 6809 assembler written in Basic for itself on itself. We both owned disk-based TRS-80 Model I systems and this seemed the natural system to write code on. We could take the generated object code and input the data to the Color Computer through DATA statements which could be POKEd into the Color Computer's memory and then CSAVEd to cassette in machine executable form. Once we could execute 6809 code on the Color Computer, we could write a serial transfer program on it and the TRS-80 Model I and send data through the RS232 ports. We decided to write the assembler in Basic for ease and speed of implementation. The following Basic program is the result.

One of the primary requirements for an assembler is an editor which can be used to write and modify source code. Rather than writing our own, we decided to use Apparat's EDTASM editor (Miosys' EDAS can also be used). 6809 source code can be input, edited and saved to disk. The "built-in" Z80 assembler is not used. The format of files saved by EDAS or EDTASM is as follows:

line 1	characters 1-7	filename (NOT USED)
line 1	characters 8-12	line number - high bit set
line 1	characters 13-	source code line text
line 2	characters 1-5	line number - high bit set
line 2	characters 6-	source code line text
line 3...n	character 1-5	line number - high bit set
...	character 6-	source code line text

End Of File (EOF) is decimal 26 (HEX 1A or control-Z)

RUNNING THE ASSEMBLER

When the 6809 assembler Basic program is run, it prompts the user for a filename. The filename must contain any extension needed - i.e "PONG/ASM". This file must have been generated by EDTASM or EDAS or be in the same format as described above. The source code file is opened and source lines are displayed on the CRT as read. After the program is read into memory, the program asks if output should be directed to the printer. If a "Y" is input, the compiled object code and associated source line is output to the printer. NOTE - the input of source code and compilation takes a substantial amount of time for long source code files. After compilation, the user can opt for DATA statement output to the printer. This allows one to copy the generated code to the Color Computer as DATA statements.

Extremely long source code files should be broken into parts as the assembler may run out of RAM on source line input or compilation. The two sample 6809 assembler listings are meant to be linked together for execution. This may be done by first POKEing the PONG program into the Color Computer and then POKEing the SETRES program. The combined programs then can be CSAVEd to tape. Later a Basic program can be RUN on the Colo Computer saving RAM for the assembled code by use of the CLEAR command. The binary code tape can be CLOADed by the Basic program and CALLed by it.

LIMITATIONS AND CHANGES FROM STANDARD 6809

Programming a 6809 assembler on the TRS-80 has forced some limitations and changes. Indirect addressing in the standard Motorola format is indicated by left and right square brackets. In this version, the exclamation point, "!" is used in place of either bracket. Also note that the standard 6809 assembler differs from a Z80 assembler in that comment fields after opcodes or labels need no ";" for delineation. An "*" in column 1 makes the entire line a comment. To assign a label to the current PC, the "*" is used - i.e. "START EQU *". Hex values begin with "$" - i.e. " ORG $E000". The Z80 pseudo op-code, DEFB (DB) corresponds to either FDB for byte definitions or FCC for string definitions - i.e. "STEP FDB $A1" and "MSG FCC /THIS IS A MESSAGE/". Generally speaking, 6809 code is PC relative. This means that generated code can be run anywhere in RAM without recompilation - no ORG statement is needed in assembler as presented. The two 6809 assembler source code files given as examples contain most of the different op-codes and pseudo op-codes used by our assembler.

Both source code and object code are kept in RAM memory. Because of this constraint, source code is limited to 270 lines and object code is limited to 1000 bytes. These constants may be adjusted in line 130 of the Basic program. Microsoft Basic is VERY slow in "string garbage collection" and makes the assembler run correspondingly slow. Five minutes for a long source code file assembly is not unrealistic (have patience!).

POSSIBLE CHANGES AND ENHANCEMENTS

The assembler as written is designed for the TRS-80 Models I & III. It should port with very minor changes to other systems running Microsoft Basic (such as CP/M based machines). To handle larger programs, the program could be easily modified so that source code lines are not kept in RAM. The program can (and has!) been slightly changed to work under the Microsoft Basic Compiler. All DIMensioned statements should be changed to include numeric constants in the declaration rather than variables as in line 140. The CLEAR statement in lines 110 and 3780 should be deleted. The passing of variable values using POKE and PEEK in line 3780 is not necessary under a compiler. With the inclusion of more logic, macros could be supported.

CONCLUDING REMARKS

The assembler as listed here is freely given to the Public Domain. We ask only acknowledgement of the authors in any distribution of the program. The program has undergone much testing and use. We are not aware of any bugs, but some may exist. Included with this article are a couple of sample programs and listings used in the construction of a Pong-type game, these also may be used and reproduced freely. We are no longer using this program in any form and no modifications or enhancements will be forthcoming.

BASIC PROGRAM LISTING

NOTE: This listing has been "formatted" for publication, so that where multiple statements (separated by colons) are found in a line, each new statement (and the colon preceding it) is placed on a separate line. Also, all continued lines (whether continued because of multiple statements on the line, or because the line was simply too long to fit the allotted column width) have been indented five spaces. When typing the program into your computer, you need not try to "format" the program as shown - it has been printed this way for clarity.

If you do not wish to type in the listings, they will be available on a TAS Public Domain Library disk. For further information on the TAS Public Domain Library, send a

self-addressed stamped envelope The Alternate Source, 704 North Pennsylvania Avenue, Lansing, Michigan 48906.

```
10 ' ********************************************
   ***************
20 ' *
                    *
30 ' *              6809 ASSEMBLER IN MICROSOFT BAS
   IC               *
40 ' *
                    *
50 ' *           BY   CLARENCE A. FELONG  &  KEN B
   ROWN             *
60 ' *           3533 PROSPECT AVE, GLENDALE, CA 9
   1214             *
70 ' *
                    *
80 ' *        DONATED TO THE PUBLIC DOMAIN - MARC
   H, 1983          *
90 ' *
                    *
100 ' ********************************************
   ***************
110 CLEAR 11500
120 DEFINT A-Z
     : DEFSNG P,T,S
130 DIM AM(270),OBJ(1000),SRC$(270),LN$(270)
140 MS=70
     : DIM SL(MS), SR(MS), SS$(MS), SV(MS)
150 ML=6
     : IN$="!"
     : VD$="+-*/, " + IN$
160 MN$(0)="/ABX/58,3/ADCA/137,8/ADCB/201,8/ADDA
     /139,8/ADDB/203,8/ADDD/195,9/ANDA/132,8/AND
     B/196,8/ANDCC/28,2/ASLA/72,3/ASLB/88,3/ASL/
     104,10/ASRA/71,3/ASRB/87,3/ASR/103,10/BCC/3
     6,5/LBCC/4132,4"
170 MN$(1)="/BCS/37,5/LBCS/4133,4/BEQ/39,5/LBEQ/
     4135,4/BGE/44,5/LBGE/4140,4/BGT/46,5/LBGT/4
     142,4/BHI/34,5/LBHI/4130,4/BHS/36,5/LBHS/41
     32,4/BITA/133,8/BITB/197,8/BLE/47,5/LBLE/41
     43,4"
180 MN$(2)="/BLO/37,5/LBLO/4133,4/BLS/35,5/LBLS/
     4131,4/BLT/45,5/LBLT/4141,4/BMI/43,5/LBMI/4
     139,4/BNE/38,5/LBNE/4134,4/BRA/32,5/LBRA/22
     ,4/BRN/33,5/LBRN/4129,4/BSR/141,5/LBSR/23,4
     /BVC/40,5/LBVC/4136,4/BVS/41,5/LBVS/4137,4"
190 MN$(3)="/CLRA/79,3/CLRB/95,3/CLR/111,10/CMPA
     /129,8/CMPB/193,8/CMPD/4227,9/CMPS/4492,9/C
     MPU/4483,9/CMPX/140,9/CMPY/4236,9/COMA/67,3
     /COMB/83,3/COM/99,10/DAA/25,3/DECA/74,3/DEC
     B/90,3/DEC/106,10"
200 MN$(4)="/EORA/136,8/EORB/200,8/EXG/30,6/INCA
     /76,3/INCB/92,3/INC/108,10/JMP/110,10/JSR/1
     73,10/LDA/134,8/LDB/198,8/LDD/204,9/LDS/430
     2,9/LDU/206,9/LDX/142,9/LDY/4238,9/LEAS/50,
     1/LEAU/51,1/LEAX/48,1/LEAY/49,1"
210 MN$(5)="/LSLA/72,3/LSLB/88,3/LSL/104,10/LSRA
     /68,3/LSRB/84,3/LSR/100,10/MUL/61,3/NEGA/64
     ,3/NEGB/80,3/NEG/96,10/NOP/18,3/ORA/138,8/O
     RB/202,8/ORCC/26,2/PSHS/52,7/PSHU/54,7"
220 MN$(6)="/PULS/53,7/PULU/55,7/ROLA/73,3/ROLB/
     89,3/ROL/105,10/RORA/70,3/RORB/86,3/ROR/102
     ,10/RTI/59,3/RTS/57,3/SBCA/130,8/SBCB/194,8
     /SEX/29,3/STA/167,10/STB/231,10/STD/237,10/
     STS/4335,10/STU/239,10"
230 MN$(7)="/STX/175,10/STY/4271,10/SUBA/128,8/S
     UBB/192,8/SUBD/131,9/SWI/63,3/SWI2/4159,3/S
     WI3/4415,3/SYNC/19,3/TFR/31,6/TSTA/77,3/TST
     B/93,3/TST/109,10
240 MN$(8)="/EQU/0,-1/FCC/0,-2/FCB/0,-3/FDB/0,-4
     "
250 DIM RP(11),EM$(14)
260 HX$="0123456789ABCDEF"
     : REG$="/D/0/X/1/Y/2/U/3/S/4/PC/5/A/8/B/9/C
     C/10/DP/11"
270 FORT=0TO11
     : READ RP(T)
     : NEXT
     : REM REG PUSH/PULL EQUATES
280 DATA6,16,32,64,64,128,0,0,2,4,1,8
290 FORT=0TO3
     : READ H(T)
     : NEXT
     : DATA 4096,256,16,1
300 REM H() = POWERS OF 16
310 EM$(0)="INVALID HEX ($HHHH) NUMBER"
320 EM$(1)="ILLEGAL LABEL"
330 EM$(2)="ILLEGAL INSTRUCTION"
340 EM$(3)="MULTIPLY DEFINED SYMBOL"
350 EM$(4)="IMPROPER STRING DELIMITER"
360 EM$(5)="UNDEFINED SYMBOL"
370 EM$(6)="MALFORMED ARITHMETIC EXPRESSION"
380 EM$(7)="OVERFLOW/UNDERFLOW"
390 EM$(8)="MISSING LABEL"
400 EM$(9)="EXTRANEOUS DATA IGNORED"
410 EM$(10)="SYNTAX ERROR (ILLEGAL ADDRESSING MO
     DE)"
420 EM$(11)="SOURCE & DESTINATION REGS INCOMPATI
     BLE"
430 EM$(12)="INVALID REGISTER DESIGNATION"
440 EM$(13)="ILLEGAL INDEX REGISTER"
450 EM$(14)="ILLEGAL USE OF INDIRECT ADDRESSING"
460 GOTO 3790
     : REM *********
470 PASS=1
     : '**** PASS 1 ****
480 PC=0
     : SN=1
490 FOR T=0 TO MS
     : SL(T)=-1
     : SR(T)=-1
     : SS$(T)=""
     : NEXT
500 SL=0
510 SL=SL+1
     : IF SL>LL THEN RETURN
     : REM *** END OF PASS 1 ***
520 CP=1
     : LB$=""
     : AM(SL)=0
     : S$=SRC$(SL)
530 IF LEN(S$)=0 THEN 510
540 IF LEFT$(S$,1) = "*" THEN 510
     : REM SKIP COMMENT LINE
550 GOSUB 1270
     : REM FIND NEXT FIELD
560 IF CP=1 THEN GOSUB 1300
     : GOSUB 1270
     : IF LB$="" THEN T=1
     : GOSUB 3540
     : CP=INSTR(S$," ")
     : GOSUB 1270
     : IF MID$(S$,CP,3)="EQU" THEN 510
570 T=INSTR(CP,S$," ")
     : IF T=0 T=LEN(S$)+1
580 T$="/" + MID$(S$,CP,T-CP) + "/"
     : LT=0
590 T=INSTR(MN$(LT),T$)
     : IF T=0 THEN LT=LT+1
     : IF LT<9 THEN 590
     : REM FIND OPCODE MNEMONIC
600 IF T=0 THEN T=2
     : GOSUB 3540
     : IF LB$="" THEN 510 ELSE TT=PC
     : GOSUB 1380
     : GOTO 510
     : REM BAD INSTR==>PUT VALUE OF LABEL AS PC
     ANYWAY
610 TT=T
     : REM SAVE PNTR TO START OF MNEMONIC STRING
620 T=INSTR(T+1,MN$(LT),"/")+1
630 CP=CP+T-TT-2
     : REM UPDATE CP TO POINT JUST PAST MNEMONIC
640 OBJ(PC)=VAL(MID$(MN$(LT),T,4))
650 AM(SL)=VAL(MID$(MN$(LT),INSTR(T,MN$(LT),",")
     +1,2))
660 IF LB$<>"" AND AM(SL)<>-1 THEN TT=PC
     : GOSUB 1380
     : REM STO VALUE OF NEWLY DEFINED LABEL AS P
```

```
     C IF THIS ISN'T AN EQU STMT
670 OPC=PC
     : REM SAVE PC VALUE FOR THIS INSTRUCTION
680 IF AM(SL)<0 THEN ON -AM(SL) GOSUB 700,770,80
    0,820 ELSE ON AM(SL) GOSUB 840,910,930,950,
    910,910,910,970,1020,1070
690 GOTO 510
700 REM "EQU" SBR
710 IF LB$="" THEN T=8
     : GOSUB 3540
     : RETURN
720 GOSUB 1270
     : EQ$=LB$
     : REM POSITION TO START OF 3RD FIELD
730 GOSUB 2750
     : IF TT=1E38 THEN RETURN
740 LB$=EQ$
     : GOSUB 1380
     : REM STORE VALUE OF LABEL
750 GOSUB 3190
760 RETURN
770 REM "FCC" SBR
780 T=INSTR(S$,"/")
     : TT=INSTR(T+1,S$,"/")
     : IF TT=0 OR T=0 THEN T=4
     : GOSUB 3540
     : AM(SL)=0
     : RETURN
     : REM FIND STRING LENGTH
790 PC = PC + TT - T - 1
     : RETURN
800 REM "FCB" SBR
810 GOSUB 1570
     : PC=PC+T
     : RETURN
820 REM "FDB" SBR
830 GOSUB 1570
     : PC=PC+T*2
     : RETURN
840 REM "AM=1" SBR
850 GOSUB 1270
     : IF MID$(S$,CP,1)=IN$ THEN CP=CP+1
     : REM GET NEXT FIELD, SKIP INDIRECT CHAR IF
      THERE
860 GOSUB 910
     : REM UPDATE PC FOR MOST INDEX CASES
870 T$=MID$(S$,CP,1)
     : IF T$="A" OR T$="B" OR T$="D" THEN IF MID
    $(S$,CP+1,1)="," THEN RETURN
     : REM INDEX MODE 5,6, & 11
880 IF T$>="A" AND T$<="Z" OR INSTR(CP,S$,",PCR"
    ) <>0 THEN PC=PC+2
     : REM INDEX MODE 9, 13
890 IF T$="," THEN RETURN
     : REM INDEX MODE 0, 1, 2, OR 3
900 T=VAL(MID$(S$,CP,6))
     : IF T=0 THEN RETURN ELSE IF T<128 AND T>-1
    29 THEN PC=PC+1
     : RETURN
     : ELSE PC=PC+2
     : RETURN
     : REM INDEX MODE 4,8, & 9
910 REM "AM=2,5,6, & 7" SBR
920 IF OBJ(PC)>255 THEN PC=PC+3
     : RETURN
     : ELSE PC=PC+2
     : RETURN
930 REM "AM=3" SBR
940 IF OBJ(PC)>255 THEN PC=PC+2
     : RETURN
     : ELSE PC=PC+1
     : RETURN
950 REM "AM=4" SBR
960 IF OBJ(PC)>255 THEN PC=PC+4
     : RETURN
     : ELSE PC=PC+3
     : RETURN
970 REM "AM=8" SBR
980 GOSUB 1270
     : GOSUB 1160
     : IF T=0 THEN RETURN
     : REM WAS EA
990 IF MID$(S$,CP,1)="#" THEN GOSUB 910
     : RETURN
     : REM IF IMMEDIATE ADDRESSING, INC PC ACCOR
    DINGLY
1000 OBJ(PC)=OBJ(PC)+32
     : REM CHANGE OBJECT CODE TO THAT FOR INDEX
    AM
1010 GOSUB 840
     : RETURN
     : REM DO PROCESSING FOR INDEX AM
1020 REM "AM=9" SBR
1030 GOSUB 1270
     : GOSUB 1160
     : IF T=0 THEN RETURN
     : REM EA
1040 IF MID$(S$,CP,1)="#" THEN GOSUB 950
     : RETURN
     : REM IF IMMEDIATE ADDRESSING, INC PC FOR D
    BL-BYTE & OP-CODE
1050 OBJ(PC)=OBJ(PC)+32
     : REM CHANGE OBJECT CODE FOR INDEX AM
1060 GOSUB 840
     : RETURN
     : REM DO INDEX AM
1070 REM AM=10 (INDEXED OR EXTENDED) PASS 1
1080 GOSUB 1270
     : GOSUB 1210
     : IF T=1 THEN GOSUB 840
     : RETURN
     : REM IF NOT EA, THEN INDEX ADDR
1090 IF MID$(S$,CP,1)<>IN$ THEN OBJ(PC)=OBJ(PC)+
    16
     : REM IF INDIR EA, OP-CODE SAME AS INDEX EA
1100 GOSUB 1110
     : RETURN
1110 REM EXTENDED ADDRESSING PASS 1 (PC UPDATE)
1120 REM ALTHO INDIRECT EXTENDED IS ACTUALLY IND
    EXED, IT IS HANDLED AS A SEPERATE SBR
1130 GOSUB 940
     : REM UPDATE FOR SIZE OF OP-CODE
1140 IF MID$(S$,CP,1)=IN$ THEN PC=PC+3 ELSE PC=P
    C+2
1150 RETURN
1160 REM FOR PASS 1, AM=8,9
     : SEE IF EXTENDED ADDRESSING, UPDATE OBJ &
    PC IF SO
1170 'OUTPUT: T=0 IF EA, T=1 IF NOT
1180 GOSUB 1210
     : IF T=1 THEN RETURN
     : REM NOT EA
1190 IF MID$(S$,CP,1)=IN$ THEN OBJ(PC)=OBJ(PC)+3
    2 ELSE OBJ(PC)=OBJ(PC)+48
     : REM UPDATE OP-CODE FOR INDIRECT OR REGULA
    R EA
1200 GOSUB 1110
     : T=0
     : RETURN
1210 REM SEE IF EXTENDED INDEXING SBR
1220 'OUTPUT: T=0 IF EXTENDED, T=1 IF NOT EXTEND
    ED
1230 IF MID$(S$,CP,1)="#" THEN T=1
     : RETURN
1240 TQ=CP
     : CP=INSTR(CP,S$," ")
     : IF CP=0 THEN CP=LEN(S$)
     : REM FIND END OF OPERAND FIELD
1250 T=INSTR(MID$(S$,TQ,CP-TQ+1),",")
     : IF T<>0 THEN T=1
     : REM IF NOT INDEXED MODE, THEN IS EXTENDED
     !!
1260 CP=TQ
     : RETURN
1270 REM "FIND NEXT FIELD" SBR
1280 IF CP=LEN(S$) THEN RETURN
1290 IF MID$(S$,CP,1)=" " THEN CP=CP+1
     : GOTO 1280
     : ELSE RETURN
1300 REM "EXTRACT LABEL" SBR--CP POINTS TO START
```

```
     OF LABEL
1310 LB$=""
1320 T$=MID$(S$,CP,1)
     : IF T$<"A" OR T$>"Z" THEN RETURN
     : REM NOT A LABEL
1330 TT=CP
     : REM FIND LENGTH OF LABEL IN TT
1340 T=INSTR(VD$,MID$(S$,TT,1))
     : IF T=0 THEN TT=TT+1
     : IF TT<=LEN(S$) THEN 1340
1350 TT=TT-CP
1360 LB$=MID$(S$,CP,TT)
     : IF TT>ML THEN LB$=LEFT$(LB$,ML)
1370 CP=CP+TT
     : RETURN
1380 REM "ADD NEW LABEL SBR"
1390 'INPUT:  LB$--LABEL
1400 '        TT--VALUE OF LABEL
1410 T$=LB$
     : GOSUB 1620
     : IF T$<>"" THEN T=3
     : GOSUB 3540
     : RETURN
     : REM FIND SPOT FOR LABEL IN SYM TBL--EXIT
     IF MD ERROR
1420 IF TT>65535 OR TT<-65536 THEN T=7
     : GOSUB 3540
     : RETURN
1430 IF LB$<SS$(LT) THEN SR(LT)=SN ELSE SL(LT)=S
     N
     : REM ATTACH NODE TO PARENT
1440 SS$(SN)=LB$
     : SV(SN)=TT
     : SN=SN+1
     : RETURN
1450 REM "GET LABEL OR NUMBER VALUE" SBR
1460 'INPUT:  CP--PNTS TO START OF LABEL/NUMBER
1470 'OUTPUT: CP--PNTS TO START OF DELIMITER
1480 '        T--VALUE OF LABEL/NUMBER--1E38 IF
     NOT DEFINED
1490 IF MID$(S$,CP,1)="*" THEN T=OPC
     : CP=CP+1
     : RETURN
     : REM IF "*" PSEUDO-OP THEN RETURN VALUE OF
      PC FOR THIS INSTR
1500 GOSUB 1300
     : IF LB$<>"" THEN T$=LB$
     : GOSUB 1620
     : IF T$="" THEN T=5
     : GOSUB 3540
     : T=1E38
     : RETURN
     : ELSE T=SV(LT)
     : RETURN
     : REM IF LABEL, FIND LABEL & RET VALUE
1510 T$=MID$(S$,CP,1)
     : IF T$="-" THEN T$=MID$(S$,CP+1,1)
1520 IF T$="$" THEN GOSUB 1730
     : RETURN
1530 IF T$<"0" OR T$>"9" THEN T=6
     : GOSUB 3540
     : T=1E38
     : RETURN
     : REM INVALID IF NOT LABEL OR NUMBER!
1540 T=VAL(MID$(S$,CP,6))
     : CP=CP+1
     : REM GET VALUE OF LABEL, SKIP CP OVER POSS
     IBLE SIGN CHAR
1550 T$=MID$(S$,CP,1)
     : IF T$>="0" AND T$<="9" THEN CP=CP+1
     : GOTO 1550
     : REM RE-ADJUST CP TO PNT TO 1ST CHAR AFTER
      DIGIT
1560 RETURN
1570 REM "COUNT COMMAS" SBR
1580 'OUTPUT: T--# OF COMMAS + 1
1590 T=0
     : GOSUB 1270
     : LT=INSTR(CP,S$," ")
     : IF LT=0 THEN LT=LEN(S$)
1600 FOR TT=CP TO LT
     : IF MID$(S$,TT,1)="," THEN T=T+1
1610 NEXT
     : T=T+1
     : RETURN
1620 REM "FIND LABEL IN SYM TABLE" SBR
1630 'INPUT:  T$--LABEL BEING SEARCHED FOR
1640 'OUTPUT:
1650 '  LABEL FOUND:  T$--LABEL AS INPUT
1660 '                LT--POINTER TO NODE WITH L
     ABEL
1670 '  LABEL NOT FOUND:  T$--NULL
1680 '                    LT--POINTER TO WOULD-B
     E PARENT NODE
1690 LT=0
1700 IF T$=SS$(LT) THEN RETURN
1710 IF T$<SS$(LT) THEN T=SR(LT) ELSE T=SL(LT)
1720 IF T=-1 THEN T$=""
     : RETURN
     : ELSE LT=T
     : GOTO 1700
1730 REM "EVALUATE '$HHHH' SBR"
1740 'INPUT: CP--POINTS TO $ WITHIN STRING
1750 'OUTPUT: CP--POINTS JUST PAST LAST 'H'
1760 '        T--$HHHH VALUE IF NO ERROR, 1E38 I
     F ERROR
1770 TT=0
     : T=0
     : CP=CP+1
1780 TQ=INSTR(HX$,MID$(S$,CP,1))
     : IF TQ=0 THEN CP=4-TT
     : T=0
     : GOSUB 3540
     : T=1E38
     : RETURN
1790 T=T+(TQ-1)*H(TT)
     : CP=CP+1
     : TT=TT+1
     : IF TT<4 THEN 1780 ELSE RETURN
1800 REM **** PASS 2 ****
1810 PASS = 2
     : PC=0
     : SL=0
1820 SL=SL+1
     : IF SL>LL THEN RETURN
     : REM *** END OF PASS 2 ***
1830 S$=SRC$(SL)
     : CP=1
1840 IF LEN(S$)=0 THEN GOSUB 3580
     : GOTO 1820
1850 IF AM(SL)=0 THEN GOSUB 3580
     : GOTO 1820
1860 T=0
     : IF MID$(S$,1,1)=" " THEN T=1
     : REM T=# OF FIELD
1870 IF MID$(S$,CP,1)=" " THEN CP=CP+1
     : GOTO 1870
     : ELSE T=T+1
     : IF T<3 THEN CP=INSTR(CP,S$," ")
     : IF CP<>0 THEN 1870
     : REM FIND START OF 3RD FIELD
1880 OPC=PC
     : IF AM(SL)>0 THEN TT=OBJ(PC)
     : IF TT>255 THEN T=2
     : GOSUB 2870
     : OBJ(PC)=T
     : OBJ(PC+1)=TT
     : PC=PC+2
     : ELSE PC=PC+1
     : REM CONVERT 2-BYTE OP-CODE TO 2 BYTES
1890 IF AM(SL)<0 THEN ON -AM(SL) GOSUB 3210,3250
     ,3290,3340 ELSE ON AM(SL) GOSUB 1940,2340,2
     390,2410,2450,2480,2540,2610,2650,2730
1900 IF OPC=PC THEN 1930 ELSE TT=OPC
     : GOSUB 2980
     : REM O/P PC
1910 IF PC-OPC > 5 THEN TT=OPC+5 ELSE TT=PC
     : REM SET LIMIT OF # OF BYTES OF OBJECT COD
     E TO O/P
1920 T=OBJ(OPC)
```

```
     : GOSUB 3060
     : GOSUB 3010
     : OPC=OPC+1
     : IF OPC<TT THEN 1920
     : REM O/P UP TO 5 BYTES OF OBJECT CODE FROM
      THIS INSTRUCTION
1930 GOSUB 3580
     : GOTO 1820
1940 REM "PASS 2--AM=1"
1950 IF MID$(S$,CP,1)=IN$ THEN GOSUB 3460
     : IF T=0 THEN RETURN
     : REM THIS  MEANS IF INDIRECT, MAY BE INDIR
     ECT EXTENDED, WHICH IS ACTUALLY INDEXED, SO
      IT'S OK HERE!!!!
1960 BPC=PC
     : REM SAVE SPOT FOR POST-BYTE
1970 PC=PC+1
     : REM SKIP SPOT FOR POST-BYTE
1980 RR=0
     : IM=0
     : PB=128
     : IF MID$(S$,CP,1)=IN$ THEN IN=1
     : CP=CP+1
     : ELSE IN=0
1990 T$=MID$(S$,CP,2)
     : IF T$="A," THEN IM=6
     : GOTO 2150
2000 IF T$="B," THEN IM=5
     : GOTO 2150
2010 IF T$="D," THEN IM=11
     : GOTO 2150
2020 IF LEFT$(T$,1)="," THEN 2170
     : REM IM=0-3
2030 REM IM=4,8,9, OR 13
2040 GOSUB 2750
     : IF T=1E38 THEN PB=0
     : GOTO 2280
2050 IO=T
2060 IF MID$(S$,CP,1) <> "," THEN T=10
     : GOSUB 3540
     : PB=0
     : GOTO 2280
2070 CP=CP+1
     : IF MID$(S$,CP,3)="PCR" THEN 2130
2080 REM IM=4,8, OR 9
     :
2090 GOSUB 3390
     : IF RR>3 THEN 2280
2100 IF IO=0 THEN IM=4
     : GOTO 2280
2110 IF IO<128 AND IO>-129 THEN IM=8
     : T=1
     : TT=IO
     : GOSUB 2870
     : OBJ(PC)=T
     : PC=PC+1
     : GOTO 2280
2120 IM=9
     : T=2
     : TT=IO
     : GOSUB 2870
     : OBJ(PC)=T
     : OBJ(PC+1)=TT
     : PC=PC+2
     : GOTO 2280
2130 REM IM=13
2140 IM=13
     : CP=CP+3
     : PC=PC+2
     : T=2
     : TT=IO-PC
     : GOSUB 2870
     : OBJ(PC-2)=T
     : OBJ(PC-1)=TT
     : GOTO 2280
2150 REM IM=5,6,11
2160 CP=CP+2
     : GOSUB 3390
     : GOTO 2280
2170 REM IM=0-3
2180 CP=CP+1
     : IF MID$(S$,CP,1)="-" THEN 2250
2190 GOSUB 3390
     : IF RR>3 THEN 2280
2200 IF MID$(S$,CP,1)<>"+" THEN T=10
     : GOSUB 3540
     : PB=0
     : GOTO 2280
2210 CP=CP+1
     : T$=MID$(S$,CP,1)
     : IF T$="" OR T$=" " OR T$=IN$ THEN 2280
     : REM DONE IF ONLY ONE "+" (IM=0)
2220 IF T$<>"+" THEN T=10
     : GOSUB 3540
     : PB=0
     : GOTO 2280
2230 CP=CP+1
     : T$=MID$(S$,CP,1)
     : IF T$<>"" AND T$<>" " AND T$<>IN$ THEN T=
     9
     : GOSUB 3540
2240 IM=1
     : GOTO 2280
2250 REM IM=2-3
2260 CP=CP+1
     : IM=2
     : IF MID$(S$,CP,1)="-" THEN IM=3
     : CP=CP+1
2270 GOSUB 3390
2280 REM SBR TO STORE POST-BYTE FOR INDEXED ADDR
     ESSING
2290 IF PB=0 THEN OBJ(BPC)=PB
     : RETURN
2300 IF IN=1 THEN IF IM=0 OR IM=2 OR MID$(S$,CP,
     1)<>IN$ THEN T=14
     : GOSUB 3540
     : OBJ(BPC)=0
     : RETURN
     : ELSE CP=CP+1
2310 GOSUB 3190
     : REM CK FOR EXTRANEOUS DATA
2320 PB=PB OR IN*16
2330 PB=PB OR RR*32
     : PB=PB OR IM
     : OBJ(BPC)=PB
     : RETURN
2340 REM "PASS 2, AM=2" SBR
2350 PC=PC+1
     : IF MID$(S$,CP,1)<>"$" THEN T=10
     : GOSUB 3540
     : T=0
     : GOTO 2380
2360 CP=CP+1
     : GOSUB 2750
     : IF TT=1E38 THEN T=0
     : GOTO 2380
2370 T=1
     : GOSUB 2870
     : REM CONVERT TO 1 "HEX" BYTE
2380 OBJ(PC-1)=T
     : GOSUB 3190
     : RETURN
2390 REM "PASS 2--AM=3" SBR
2400 RETURN
     : REM NOTHING TO DO!
2410 REM "PASS 2--AM=4" SBR
2420 PC=PC+2
     : GOSUB 2750
     : IF TT=1E38 THEN TT=0 ELSE TT=TT-PC
2430 T=2
     : GOSUB 2870
     : OBJ(PC-2)=T
     : OBJ(PC-1)=TT
2440 GOSUB 3190
     : RETURN
2450 REM "PASS 2--AM=5" SBR
2460 PC=PC+1
     : GOSUB 2750
     : IF TT=1E38 THEN TT=0 ELSE TT=TT-PC
2470 T=1
```

```
     : GOSUB 2870
     : OBJ(PC-1)=T
     : GOSUB 3190
     : RETURN
2480 REM "PASS 2--AM=6" SBR
2490 PC=PC+1
     : PB=0
     : GOSUB 3100
     : IF T>11 THEN 2530
2500 IF MID$(S$,CP,1) <> "," THEN T=10
     : GOSUB 3540
     : GOTO 2530
2510 PB=T
     : CP=CP+1
     : GOSUB 3100
     : IF T>11 THEN PB=0
     : GOTO 2530
2520 GOSUB 3190
     : IF (PB>5 AND T<6) OR (PB<6 AND T>5) THEN
     PB=0
     : T=11
     : GOSUB 3540
     : ELSE PB=PB+16*T
     : REM REGS MUST BE = LEN
2530 OBJ(PC-1)=PB
     : RETURN
2540 REM "PASS 2--AM=7" SBR
2550 PC=PC+1
     : PB=0
2560 GOSUB 3100
     : IF T>11 THEN STOPPB=0
     : GOTO 2600
2570 IF (T=3 AND OBJ(PC-2)>53) OR (T=4 AND OBJ(P
     C-2)<54) THEN PB=0
     : T=12
     : GOSUB 3540
     : GOTO 2600
     : REM CK THAT U/S REGS OK
2580 PB=PB OR RP(T)
2590 IF MID$(S$,CP,1)="," CP=CP+1
     : GOTO 2560 ELSE GOSUB 3190
2600 OBJ(PC-1)=PB
     : RETURN
2610 REM "PASS 2--AM=8" SBR
2620 IF MID$(S$,CP,1)="#" THEN GOSUB 2340
     : RETURN
2630 GOSUB 3460
     : IF T=1 THEN GOSUB 1940
     : REM IF NOT EA, THEN INDEXED
2640 RETURN
2650 REM "PASS 2--AM=9" SBR
2660 IF MID$(S$,CP,1)="#" THEN 2690
2670 GOSUB 3460
     : IF T=1 THEN 1940
     : REM IF NOT EA, THEN INDEXED
2680 RETURN
2690 REM AM=2-BYTE IMMEDIATE ADDRESSING
     :
2700 PC=PC+2
     : CP=CP+1
     : GOSUB 2750
     : IF TT=1E38 THEN TT=0
2710 T=2
     : GOSUB 2870
2720 OBJ(PC-2)=T
     : OBJ(PC-1)=TT
     : GOSUB 3190
     : RETURN
2730 REM "PASS 2--AM=10 (INDEXED OR EXTENDED)" S
     BR
2740 GOSUB 3460
     : IF T=0 THEN RETURN ELSE GOSUB 1940
     : RETURN
     : REM IF NOT EXTENDED, THEN INDEXED
2750 REM "EVALUATE EXPRESSION" SUBROUTINE
2760 'INPUT:  CP--POINTS TO START OF EXPRESSION
2770 'OUTPUT: CP--POINTS TO EXPRESSION DELIMITER
      IF NO ERROR
2780 '         TT--EXPRESSION VALUE
2790 '           --1E38 IF ERROR
2800 '
2810 GOSUB 1450
     : IF T=1E38 THEN TT=T
     : RETURN
     : ELSE TQ=T
     : REM GET VALUE OF 1ST (ONLY?) OPERAND OF E
     XPRESSION
2820 OP$=MID$(S$,CP,1)
     : IF INSTR(VD$,OP$)=0 AND OP$<>"" THEN T=6
     : GOSUB 3540
     : TT=1E38
     : RETURN
     : REM IF OPERATOR INVALID, RETURN
2830 IF INSTR(" ,"+IN$,OP$)<>0 OR OP$="" THEN TT
     =TQ
     : RETURN
     : REM DONE IF "OPERATOR" IS A VALID EXPRESS
     ION DELIMITER
2840 CP=CP+1
     : GOSUB 1450
     : IF T=1E38 THEN TT=T
     : RETURN
2850 IF OP$="+" THEN TQ=TQ+T ELSE IF OP$="-" THE
     N TQ=TQ-T ELSE IF OP$="*" THEN TQ=TQ*T ELSE
      TQ=TQ/T
2860 GOTO 2820
2870 REM CONVERT 1 DECIMAL NUMBER TO 1 OR 2 "HEX
     " BYTES (>0)
2880 'INPUT:  T=# OF BYTES
2890 '        TT=NUMBER
2900 'OUTPUT: T=1ST BYTE
2910 '        TT=2ND BYTE (IF NEEDED)
2920 IF T=2 THEN 2950
2930 IF TT>255 OR TT<-256 OR ((TT> 127 OR TT<-12
     8) AND AM(SL)=5) THEN T=7
     : GOSUB 3540
     : T=0
     : RETURN
2940 IF TT<0 THEN T=TT+256
     : RETURN
     : ELSE T=TT
     : RETURN
2950 IF TT>65535 OR TT<-65536 THEN T=7
     : GOSUB 3540
     : T=0
     : TT=0
     : RETURN
2960 IF TT<0 THEN TT=TT+65536
2970 T=INT(TT/256)
     : TT=TT-T*256
     : RETURN
2980 REM OUTPUT 2 HEX BYTES, FOLLOWED BY A SPACE
2990 'INPUT: TT--NUMBER IN DECIMAL FORM (EG, PC)
3000 T=2
     : GOSUB 2870
     : GOSUB 3060
     : GOSUB 3010
     : T=TT
     : GOSUB 3060
     : GOSUB 3010
     : T$=" "
     : GOSUB 3010
     : RETURN
3010 REM PRINT T$ ON CRT AND POSSIBLY LP (IF PF=
     1)
3020 'INPUT: T$--STRING TO PRINT
3030 '       PF--1 IF O/P TO PRINTER ENABLED
3040 PRINT T$;
     : IF PF=1 THEN LPRINT T$;
3050 RETURN
3060 REM CONVERT "HEX" (>0) BYTE TO ASCII HEX ST
     RING
3070 'INPUT:  T--"HEX" BYTE
3080 'OUTPUT: T$--HEX ASCII BYTE
3090 T$=MID$(HX$,INT(T/16)+1,1) + MID$(HX$,T-(IN
     T(T/16)*16)+1,1)
     : RETURN
3100 REM SBR TO GET REGISTER FROM INPUT STRING
3110 'INPUT:  CP--POINTS TO A REGISTER IN I/P ST
     RING
```

```
3120 'OUTPUT: CP--POINTS TO CHAR AFTER REGISTER
     IF ONE FOUND
3130 '         T--THE # OF REGISTER, IF ONE FOUND
3140 '          -- >11 IF NO VALID REGISTER FOUND
3150 IF LEN(S$)=CP THEN 3170
     : REM CAN'T BE A 2-CHAR REG
3160 T$="/" + MID$(S$,CP,2) + "/"
     : T=INSTR(REG$,T$)
     : IF T<>0 THEN LT=2
     : GOTO 3180
     : REM LOOK FOR 2-CHAR REG 1ST
3170 T$="/" + MID$(S$,CP,1) +"/"
     : T=INSTR(REG$,T$)
     : IF T=0 THEN T=12
     : GOSUB 3540
     : RETURN
     : ELSE LT=1
3180 T=VAL(MID$(REG$,T+LT+2,LT))
     : CP=CP+LT
     : RETURN
3190 T$=MID$(S$,CP,1)
     : IF T$<>"" AND T$<>" " THEN T=9
     : GOSUB 3540
3200 RETURN
     : REM CHECK FOR EXTRANEOUS DATA SBR
3210 REM "PASS 2--EQU" SBR
3220 CP=1
     : GOSUB 1300
     : IF LB$="" THEN T$="???? "
     : GOSUB 3010
     : RETURN
3230 T$=LB$
     : GOSUB 1620
     : IF T$="" THEN T$="???? "
     : GOSUB 3010
     : RETURN
3240 TT=SV(LT)
     : GOSUB 2980
     : RETURN
3250 REM "PASS 2--FCC" SBR
3260 CP=CP+1
3270 T$=MID$(S$,CP,1)
     : IF T$<>"/" THEN OBJ(PC)=ASC(T$)
     : PC=PC+1
     : CP=CP+1
     : GOTO 3270
3280 RETURN
3290 REM "PASS 2--FCB" SBR
3300 GOSUB 2750
     : IF TT=1E38 THEN TT=0
3310 T=1
     : GOSUB 2870
     : REM CONVERT TO A 1-BYTE NUMBER
3320 OBJ(PC)=T
     : PC=PC+1
     : T$=MID$(S$,CP,1)
     : IF T$<>"," THEN GOSUB 3190
     : RETURN
     : REM DONE AFTER LAST COMMA FOLLOWED BY A #
3330 CP=CP+1
     : IF MID$(S$,CP,1)="," THEN 3320
     : ELSE 3300
3340 REM "PASS 2--FDB" SBR
3350 GOSUB 2750
     : IF TT=1E38 THEN TT=0
3360 T=2
     : GOSUB 2870
     : REM CONVERT TO A 2-BYTE NUMBER
3370 OBJ(PC)=T
     : OBJ(PC+1)=TT
     : PC=PC+2
     : T$=MID$(S$,CP,1)
     : IF T$<>"," THEN GOSUB 3190
     : RETURN
3380 CP=CP+1
     : IF MID$(S$,CP,1)="," THEN 3370
     : ELSE 3350
3390 REM "GET INDEX REG" SBR
3400 'INPUT:  CP--POINTS TO "R"
3410 'OUTPUT: CP--POINTS TO JUST AFTER "R"
3420 '        RR--0-3 IF VALID REG, >3 IF NOT.
3430 GOSUB 3100
     : RR=T-1
     : IF RR<0 OR RR>3 THEN STOP
     : T=13
     : R=13
     : GOSUB 3540
     : PB=0
     : RETURN
3440 T$=MID$(S$,CP,1)
     : IF T$<>"" AND T$<>" " AND T$<>"+" AND T$<
     >IN$ THEN T=9
     : GOSUB 3540
3450 RETURN
3460 REM "PASS 2 EXTENDED" SBR
3470 'OUTPUT: T=1 IF NOT EXTENDED ADDRESSING
3480 '        T=0 IF EA.  IF EA, PC, OBJ(PC) UPD
     ATED
3490 GOSUB 1210
     : IF T=1 THEN RETURN
3500 IF MID$(S$,CP,1)=IN$ THEN IN=1
     : CP=CP+1
     : ELSE IN=0
3510 GOSUB 2750
     : IF TT=1E38 THEN TT=0
3520 IF IN=1 THEN OBJ(PC)=159
     : PC=PC+1
     : IF MID$(S$,CP,1)<>IN$ THEN T=14
     : GOSUB 3540
     : REM IF INDIRECT ADDRESSING, ACTUALLY IS I
     NDEXED ADDRESSING, SO STORE POSTBYTE = #9F
3530 T=2
     : GOSUB 2870
     : OBJ(PC)=T
     : OBJ(PC+1)=TT
     : PC=PC+2
     : T=0
     : RETURN
3540 PRINT"PASS"PASS"ERROR
     : ";EM$(T)
3550 IF PF=1 THEN LPRINT "PASS"PASS"ERROR
     : ";EM$(T)
3560 IF PASS=1 THEN GOSUB 3580
3570 RETURN
3580 REM PRINT LINE NUMBER & SOURCE LINE SBR
3590 PRINT TAB(16) LN$(SL);
     : IF PF=1 THEN LPRINT TAB(16) LN$(SL);
3600 PRINT TAB(22) S$
     : IF PF=1 THEN LPRINT TAB(22) S$
3610 RETURN
3620 PRINT
     : PRINT"CURRENT TIME AND DATE ARE
     : "TIME$
3630 PRINT
     : INPUT"DO YOU WANT LINE PRINTER OUTPUT?";T
     $
     : IF T$="Y" THEN PF=1 ELSE PF=0
3640 IF PF=1 THEN LPRINT"6809 ASSEMBLER  -  VERS
     ION 1.2                                  ";TIME
     $
     : LPRINT" "
     : LPRINT " "
3650 GOSUB 470
     : GOSUB 1800
3660 PRINT"DUMP OF LABEL TABLE FOLLOWS
     :"
3670 FORT=1TOSN-1
     :PRINTSS$(T),SV(T)
     :NEXT
3680 CLS
     : INPUT"DO YOU WANT TO
     :

(1) SEND OBJECT CODE TO PRINTER
(2) RETURN TO EDTASM
(3) RETURN TO DOS
(4) RE-ASSEMBLE

";T
3690 IF T<1 OR T>4 THEN 3680 ELSE IF T=4 THEN 36
     20
```

```
3700 IF T<>1 THEN 3780
3710 INPUT"STARTING LINE NUMBER?";LN
     :Z=0
3720 LPRINTSTR$(LN);" DATA";STR$(PC);"
     ;REM==>SIZE OF OBJECT CODE"
3730 LN=LN+10
 740 LPRINT STR$(LN);" DATA";STR$(OBJ(Z));
     :T=0
3750 Z=Z+1
     :IFZ>PC-1THEN3770 ELSE LPRINT",";MID$(STR$(
     OBJ(Z)),2);
3760 T=T+1
     :IFT<7THEN3750ELSE
     :LPRINT
     :Z=Z+1
     :IFZ<=PC-1THEN3730
3770 GOTO3680
3780 POKE -200,T
     : CLEAR 50
     : T=PEEK(-200)
     : IF T=2 THEN CMD"EDTASM" ELSE CMD"S"
3790 CLS
3800 I=1
3810 INPUT"ENTER FILE TO BE INPUT ";FI$
3820 OPEN"I",1,FI$
3830 LINEINPUT#1,B$
3840 IP$=RIGHT$(B$,LEN(B$)-7)
3850 GOSUB 3940
3860 PRINTLN$(I);" ";SR$(I)
3870 I=I+1
3880 LINEINPUT#1,IP$
3890 IF EOF(1) OR LEFT$(IP$,1)=CHR$(26) THEN 393
     0
3900 GOSUB 3940
3910 PRINTLN$(I);" ";SR$(I)
3920 GOTO 3870
3930 LL=I-1
     : GOTO 3620
3940 FOR J=1 TO 5
3950 HO$=HO$+CHR$(ASC(MID$(IP$,J,1))-128)
 960 NEXT
 970 HO=0
     :SP=0
3980 LN$(I)=HO$
3990 HO$=""
4000 SR$(I)=RIGHT$(IP$,LEN(IP$)-6)
4010 HO=INSTR(SR$(I),CHR$(9))
4020 IF HO=0 THEN 4060
4030 SP=HO-(INT(HO/8)*8)
     :SP=8-SP
4040 SR$(I)=LEFT$(SR$(I),HO-1)+STRING$(SP," ")+R
     IGHT$(SR$(I),LEN(SR$(I))-HO)
4050 GOTO 4010
4060 RETURN
```

VARIABLE USAGE FOR 6809 ASSEMBLER

AM()	ADDRESS MODES OF OPCODES - IMMEDIATE, INDIRECT, ETC.
B$	STRING FOR INPUT LINE FROM SOURCE FILE
BPC	TEMPORARY PROGRAM COUNTER STORAGE - USED W/ POST BYTE
CP	CHARACTER POINTER USED WHEN PARSING INPUT SOURCE LINE
EM$()	ERROR MESSAGE STRING ARRAY
EQ$	TEMPORARY HOLD FOR LABELS
FI$	FILE NAME FOR SOURCE INPUT
H()	ARRAY OF POWERS OF 16 FOR DECIMAL TO HEX CONVERSIONS
HO	USED FOR POSITION FORMATTING FOR OBJECT CODE AND SYMBOL TABLE
HO$	STRING USED FOR STRING FORMATTING (SEE HO)
HX$	HEX CHARACTERS STRING USED FOR DISPLAYING HEX VALUES
I	INDEX FOR NUMBER OF INPUT SOURCE CODE LINES
IM	INPUT MODE (TYPE OF ADDRESSING) USED FOR BYTE COUNT, ETC.
IN	INDIRECT ADDRESSING MODE FLAG
IN$	INDIRECT ADDRESSING MODE CHARACTER - ON TRS-80, "!"
IO	TEMPORARY HOLD FOR TYPE (T) OF OP-CODE
[illegible]$	INPUT STRING FOR SOURCE CODE LINE NUMBER AS SET BY EDTASM OR EDAS
	TEMPORARY LOOP COUNTER
L$	STRING REPRESENTATION OF PROGRAM LABEL
LL	LINE COUNT LIMIT (NUMBER OF INPUT LINES)
LN	START LINE NUMBER FOR DATA STATEMENT OUTPUT
LN$()	LINE NUMBER STRING ARRAY USED FOR ASSEMBLY LISTINGS
LT	POINTER TO SYMBOL TABLE NODE FOR GIVEN LABEL
ML	MAXIMUM LABEL LENGTH (6 ON TRS-80)
MN$()	OPCODE MNEMONIC STRING ARRAY TABLE
MS	MAXIMUM SIZE FOR FOR SYMBOL TABLE NODE ARRAYS
OBJ()	ARRAY FOR HOLDING GENERATED OBJECT CODE
OPC	TEMPORARY HOLD FOR SINGLE INSTRUCTION OP-CODE VALUE
OP$	STRING TO HOLD OPERATOR (+,-,/,*)
PASS	VARIABLE FOR ASSEMBLER PASS NUMBER
PB	POST BYTE VALUE USED FOR CERTAIN OP-CODES
PC	PROGRAM COUNTER (PC) VALUE
PF	PRINT FLAG (VALUE = 1 MEANS OUTPUT TO PRINTER)
REG$	REGISTER SYMBOLS AND NUMERIC ASSIGNMENT STRING
RP()	REGISTER PUSH/PULL EQUATES ARRAY
RR	REGISTER VALUE RANGE (0-3 VALID RANGE)
S$	TEMPORARY STRING FOR SOURCE CODE LINE
SL	SOURCE LINE COUNT FOR PROCESSING PASSES
SL()	LEFT SIDE VALUES OF BALANCED TREE FOR SYMBOL TABLE
SN	SYMBOL NUMBER (SYMBOL TABLE COUNT/INDEX)
SP	SPACE COUNT USED FOR FORMATTING OUTPUT
SR()	RIGHT SIDE VALUES OF BALANCED TREE FOR SYMBOL TABLE
SRC$()	SOURCE CODE LINE HOLD ARRAY
SS$()	SYMBOL TABLE STRING ARRAY
SV()	SYMBOL TABLE VALUE ARRAY
T	TEMPORARY VARIABLE
T$	TEMPORARY STRING VARIABLE
TO	TEMPORARY VARIABLE WHEN T ALREADY IN USE
TT	VARIABLE USED TO HOLD DECIMAL VALUE OF PC
VD$	VALID DELIMITERS STRING
Z	LENGTH OF CODE FOR USE WITH DATA STATEMENTS

SAMPLE LISTING #1 - PONG/ASM (ASSEMBLER OUTPUT)

```
6809 ASSEMBLER  -  VERSION 1.2                              03/01/83 18:07:32


0155            00100 LOC1   EQU    341             KEYBOARD SCAN LOCS
0156            00110 LOC2   EQU    342
0158            00120 LOC3   EQU    344
0153            00130 LOC4   EQU    339
0E74            00140 NOPLAY EQU    3700            AREA ABOVE BASIC PROG
0E75            00150 XBALL  EQU    NOPLAY+1
0E76            00160 YBALL  EQU    XBALL+1
0E77            00170 DELTAX EQU    YBALL+1
0E78            00180 DELTAY EQU    DELTAX+1
0E79            00190 LFTPDL EQU    DELTAY+1
0E7A            00200 RGTPDL EQU    LFTPDL+1
0E7B            00210 LFTCLR EQU    RGTPDL+1
0E7C            00220 RGTCLR EQU    LFTCLR+1
0E7D            00230 STATUS EQU    RGTCLR+1
0E7E            00240 XCOORD EQU    STATUS+1
0E7F            00250 YCOORD EQU    XCOORD+1
0E80            00260 COLOR  EQU    YCOORD+1
0E81            00270 OPTION EQU    COLOR+1
0E82            00280 RESULT EQU    OPTION+1
0E83            00290 DOTBIT EQU    RESULT+1
0000 3437       00300        PSHS   D,X,Y,CC        SAVE REGISTERS
0002 BD1D       00310        BSR    MVPDL1          MOVE LEFT PADDLE
0004 B60E74     00320        LDA    NOPLAY          GET NUMBER OF PLAYERS
0007 8102       00330        CMPA   #2              2 PLAYERS?
0009 2602       00340        BNE    AHEAD1          NO - CONTINUE
000B BD7B       00350        BSR    MVPDL2          YES - MOVE RIGHT PADDLE
000D 7F0E7D     00360 AHEAD1 CLR    STATUS          CLEAR STATUS
0010 1700DC     00370        LBSR   MVBALL          MOVE BALL - SET STATUS
0013 BD0C       00380        BSR    MVPDL1          MOVE PADDLE 1
0015 B60E74     00390        LDA    NOPLAY          NO. OF PLAYERS ?
0018 8102       00400        CMPA   #2
001A 2602       00410        BNE    GETOUT          1 - CONT
001C BD6A       00420        BSR    MVPDL2          2 - MOVE PADDLE
001E 3537       00430 GETOUT PULS   D,CC,X,Y
0020 39         00440        RTS                    RETURN TO BASIC PROGRAM
                00450 *
                00460 *      MOVE LEFT PADDLE SUBROUTINE
                00470 *
0021 86FF       00480 MVPDL1 LDA    #$00FF          RESET KEYIN BITS FOR NEW
0023 B70155     00490        STA    LOC1                  KEYBOARD READ
0026 B70156     00500        STA    LOC2
0029 8601       00510        LDA    #1              LEFT PADDLE X - COORD
002B B70E7E     00520        STA    XCOORD          SAVE FOR SUBRTN CALL
002E B60E7B     00530        LDA    LFTCLR          LEFT PADDLE COLOR
```

```
0031 B70E80   00980          STA   COLOR          STORE FOR SUBRTN CALL
0034 7F0E81   01000          CLR   OPTION         0 = SET OPTION
0037 AD9FA000 01020          JSR   '$A000'        POLL KEYBOARD
003B 815E     01040          CMPA  #$005E         UP ARROW ?
003D 2623     01060          BNE   AHEAD2         NO - CONTINUE
003F B60E79   01080          LDA   LFTPDL         GET CURENT PADDLE POS.
0042 8103     01100          CMPA  #3             UPPER BOUND ?
0044 2741     01120          BEQ   PD1OUT         YES - EXIT
0046 8001     01140          SUBA  #1             NEW = OLD - 1
0048 B70E79   01160          STA   LFTPDL         SAVE NEW POSITION
004B 8001     01180          SUBA  #1             A = NEW POS TO SET
004D B70E7F   01200          STA   YCOORD         STORE Y FOR SUBRTN CALL
0050 170147   01220          LBSR  SRPSBR         SET NEW POINT
0053 8B03     01240          ADDA  #3             A = OLD POS TO RESET
0055 B70E7F   01260          STA   YCOORD         STORE Y FOR SUBRTN CALL
0058 C601     01280          LDB   #1             1 = RESET
005A F70E81   01300          STB   OPTION         SET UP CALL
005D 17013A   01320          LBSR  SRPSBR         RESET OLD POINT
0060 2025     01340          BRA   PD1OUT         EXIT
0062 810A     01360 AHEAD2   CMPA  #$000A         DOWN ARROW ?
0064 2621     01380          BNE   PD1OUT         NO - EXIT
0066 B60E79   01400          LDA   LFTPDL         GET LEFT PADDLE POS
0069 811C     01420          CMPA  #28            LAST POSITION ?
006B 271A     01440          BEQ   PD1OUT         YES - EXIT
006D 8B01     01460          ADDA  #1             POINT TO NEW POS
006F B70E79   01480          STA   LFTPDL         STORE NEW POS
0072 8B01     01500          ADDA  #1             NEW POINT TO SET
0074 B70E7F   01520          STA   YCOORD         SAVE FOR SUBRTN
0077 170120   01540          LBSR  SRPSBR         TURN ON POINT
007A 8003     01560          SUBA  #3             OLD POINT TO RESET
007C B70E7F   01580          STA   YCOORD         SAVE FOR SUBRTN
007F 8601     01600          LDA   #1             OPTION = RESET
0081 B70E81   01620          STA   OPTION
0084 170113   01640          LBSR  SRPSBR         RESET OLD POINT
0087 39       01660 PD1OUT   RTS                  EXIT
              01680 *
              01700 *        MOVE RIGHT PADDLE SUBROUTINE
              01720 *
0088 86FF     01740 MVPDL2   LDA   #$00FF         RESET CURRENT KEY-IN
008A B70158   01760          STA   LOC3                 FOR NEW READ
008D B70153   01780          STA   LOC4
0090 863E     01800          LDA   #62            COLUMN = 62 FOR RIGHT
0092 B70E7E   01820          STA   XCOORD         SAVE FOR SUBRTN CALL
0095 B60E7C   01840          LDA   RGTCLR         GET RIGHT PADDLE COLOR
0098 B70E80   01860          STA   COLOR          STORE FOR SUBRTN CALL
009B 7F0E81   01880          CLR   OPTION         OPTION = SET
009E AD9FA000 01900          JSR   '$A000'        POLL KEYBOARD
00A2 8109     01920          CMPA  #9             LEFT ARROW ?
00A4 2623     01940          BNE   AHEAD3         NO - CONTINUE
00A6 B60E7A   01960          LDA   RGTPDL         GET RIGHT POSITION
00A9 8103     01980          CMPA  #3             UPPER LIMIT ?
00AB 2741     02000          BEQ   PD2OUT         YES - EXIT
00AD 8001     02020          SUBA  #1             NEW POS = OLD - 1
00AF B70E7A   02040          STA   RGTPDL         STORE NEW POSITION
00B2 8001     02060          SUBA  #1             POINT TO SET
00B4 B70E7F   02080          STA   YCOORD         SAVE FOR SUBRTN CALL
00B7 1700E0   02100          LBSR  SRPSBR         SET NEW POINT
00BA 8B03     02120          ADDA  #3             POINT TO RESET
00BC B70E7F   02140          STA   YCOORD         SAVE IT
00BF C601     02160          LDB   #1             1 = RESET
00C1 F70E81   02180          STB   OPTION         OPTION = RESET
00C4 1700D3   02200          LBSR  SRPSBR         RESET OLD POINT
00C7 2025     02220          BRA   PD2OUT         EXIT
00C9 810C     02240 AHEAD3   CMPA  #$000C         CLEAR KEY ?
00CB 2621     02260          BNE   PD2OUT         NO - EXIT
00CD B60E7A   02280          LDA   RGTPDL         GET RIGHT POS
00D0 811C     02300          CMPA  #28            LOWER LIMIT ?
00D2 271A     02320          BEQ   PD2OUT         YES - EXIT
00D4 8B01     02340          ADDA  #1             NEW POS = OLD POS + 1
00D6 B70E7A   02360          STA   RGTPDL         SAVE NEW POS
00D9 8B01     02380          ADDA  #1             NEW POINT TO SET
00DB B70E7F   02400          STA   YCOORD         SAVE FOR SUBRTN CALL
00DE 1700B9   02420          LBSR  SRPSBR         SET NEW POINT
00E1 8003     02440          SUBA  #3             POINT TO RESET
00E3 B70E7F   02460          STA   YCOORD         SAVE IT
00E6 C601     02480          LDB   #1             1 = RESET
00E8 F70E81   02500          STB   OPTION         OPTION = RESET
00EB 1700AC   02520          LBSR  SRPSBR         RESET POINT
00EE 39       02540 PD2OUT   RTS                  EXIT
              02560 *
              02580 *        MOVE BALL SUBROUTINE
              02600 *
00EF 8601     02620 MVBALL   LDA   #1             OPTION = RESET
00F1 B70E81   02640          STA   OPTION
00F4 8605     02660          LDA   #5             COLOR = WHITE
00F6 B70E80   02680          STA   COLOR
00F9 B60E75   02700          LDA   XBALL          GET BALL X-COORD
00FC B70E7E   02720          STA   XCOORD         STORE IT
00FF F60E76   02740          LDB   YBALL          GET BALL Y-COORD
0102 F70E7F   02760          STB   YCOORD         STORE IT
0105 170092   02780          LBSR  SRPSBR         RESET OLD BALL
0108 BB0E77   02800          ADDA  DELTAX         GET NEW X-POS
010B FB0E78   02820          ADDB  DELTAY         GET NEW Y-POS
010E B70E75   02840 SKIP1    STA   XBALL          STORE NEW X COORD
0111 B70E7E   02860          STA   XCOORD
0114 C102     02880          CMPB  #2
0116 2C02     02900          BGE   SKIP2          >= 2 - SKIP
0118 C602     02920          LDB   #2
011A C11D     02940 SKIP2    CMPB  #29
011C 2F02     02960          BLE   SKIP3          <= 28 - SKIP
011E C61D     02980          LDB   #29
0120 F70E76   03000 SKIP3    STB   YBALL          STORE NEW Y VALUE
0123 F70E7F   03020          STB   YCOORD
0126 7F0E81   03040          CLR   OPTION         OPTION = SET
0129 17006E   03060          LBSR  SRPSBR         SET NEW BALL
012C 8102     03080          CMPA  #2             LEFT BORDER ?
012E 2605     03100          BNE   BALL1          NO - CHECK RIGHT
0130 F00E79   03120          SUBB  LFTPDL         PADDLE POS - Y COORD
0133 201B     03140          BRA   BALL2          JUMP AROUND RIGHT LOGIC
0135 813D     03160 BALL1    CMPA  #61            RIGHT BORDER ?
0137 264E     03180          BNE   BALL4          NO - MISSED BALL
0139 3402     03200          PSHS  A              SAVE A
013B B60E74   03220          LDA   NOPLAY         2 PLAYERS ?
013E 8102     03240          CMPA  #2
0140 3502     03260          PULS  A              RESTORE A
0142 2709     03280          BEQ   PATCH1         2 - CONT
0144 C601     03300          LDB   #1
0146 F70E7D   03320          STB   STATUS
0149 C610     03340          LDB   #16
0148 2006     03360          BRA   PATCH2         1 PLAYER
014D F00E7A   03380 PATCH1   SUBB  RGTPDL         BALL & PADDLE POS
0150 F70E78   03400 BALL2    STB   DELTAY         IF HIT - NEW DELTA Y
0153 700E77   03420 PATCH2   NEG   DELTAX         DELTA X = - DELTA X
0156 C1FF     03440          CMPB  #$00FF         MINUS ONE ?
0158 2726     03460          BEQ   BALL3          YES - STATUS = HIT
015A C100     03480          CMPB  #0             0 ?
015C 2722     03500          BEQ   BALL3          YES - STATUS = HIT
015E C101     03520          CMPB  #1             1 ?
0160 271E     03540          BEQ   BALL3          YES - STATUS = HIT
0162 813D     03560          CMPA  #61            RIGHT LIMIT ?
0164 2613     03580          BNE   BALL2B         NO - STATUS = MISSED
0166 B60E74   03600          LDA   NOPLAY         GET NO. OF PLAYERS
0169 8102     03620          CMPA  #2             2 ?
016B 2705     03640          BEQ   BALL2A         YES - CHECK HIT/MISS
016D 700E77   03660          NEG   DELTAX         NO - DELTAX= -DELTAX
0170 2015     03680          BRA   BALL4          CHECK BORDERS
0172 8604     03700 BALL2A   LDA   #4             STATUS = RIGHT MISS
0174 B70E7D   03720          STA   STATUS
0177 2020     03740          BRA   BALOUT         EXIT
0179 8603     03760 BALL2B   LDA   #3             STATUS = LEFT MISS
017B B70E7D   03780          STA   STATUS
017E 2019     03800          BRA   BALOUT         EXIT
0180 B602     03820 BALL3    LDA   #2             STATUS = PADDLE HIT
0182 B70E7D   03840          STA   STATUS
0185 2012     03860          BRA   BALOUT         EXIT
0187 C102     03880 BALL4    CMPB  #2             CHECK UPPER BOUND
0189 2F06     03900          BLE   BALL5          YES - SET STATUS
018B C11D     03920          CMPB  #29            CHECK LOWER BOUND
018D 2C02     03940          BGE   BALL5          YES - SET STATUS
018F 2008     03960          BRA   BALOUT         GET OUT
0191 700E78   03980 BALL5    NEG   DELTAY         DELTA Y = - DELTA Y
0194 8601     04000          LDA   #1             STATUS = BOUNDRY HIT
0196 B70E7D   04020          STA   STATUS
0199 39       04040 BALOUT   RTS                  EXIT
019A          04060 SRPSBR   EQU   *              ADD CODE FOR GRAPHICS
```

SAMPLE LISTING #1 STARTS ON NEXT PAGE

SAMPLE LISTING #2 - SETRES/ASM (ASSEMBLER OUTPUT)

```
6809 ASSEMBLER  -  VERSION 1.2                          03/01/83 20:18:39

                        00100 *
                        00120 * SET/RESET/POINT FOR LOW-RES COLOR GRAPHICS
                        00140 *
                        00160 * FOR USE WITH PONG - NOTE EQUATES
                        00180 *
                        00220 *
0E74                    00240 NOPLAY EQU    3700
0E75                    00260 XBALL  EQU    NOPLAY+1
0E76                    00280 YBALL  EQU    XBALL+1
0E77                    00300 DELTAX EQU    YBALL+1
0E78                    00320 DELTAY EQU    DELTAX+1
0E79                    00340 LFTPDL EQU    DELTAY+1
0E7A                    00360 RGTPDL EQU    LFTPDL+1
0E7B                    00380 LFTCLR EQU    RGTPDL+1
0E7C                    00400 RGTCLR EQU    LFTCLR+1
0E7D                    00420 STATUS EQU    RGTCLR+1
0E7E                    00440 XCOORD EQU    STATUS+1
0E7F                    00460 YCOORD EQU    XCOORD+1
0E80                    00480 COLOR  EQU    YCOORD+1
0E81                    00500 OPTION EQU    COLOR+1
0E82                    00520 RESULT EQU    OPTION+1
0E83                    00540 DOTBIT EQU    RESULT+1
                        00560 *
                        00580 *   SRPSBR--SET, RESET, POINT SBR
                        00600 *   INPUT:  XCOORD--# FROM 0 - 63
                        00620 *           YCOORD---# FROM 0 - 31
                        00640 *           COLOR----# FROM 1 - 8 IF OPTION = 0
                        00660 *           OPTION---0=SET, 1=RESET, 2=POINT
                        00680 *   OUTPUT: RESULT--AS IN BASIC FN POINT(X,Y)
                        00700 *
                        00720 *   USES FOLLOWING ALGORITHM TO DETERMINE DOT
                        00740 *
                        00760 *         X  Y   XLATION    RESULTANT #
                        00780 *         E  E    0  0  (0)     8
                        00800 *         D  E    0  1  (1)     4
                        00820 *         E  D    1  0  (2)     2
                        00840 *         D  D    1  1  (3)     1
                        00860 *          (E=EVEN, O=ODD)
                        00880 *
0000 3437               00900 SRPSBR PSHS   D,X,Y,CC        SAVE IMPORTANT REGS
0002 5F                 00920        CLRB                   B WILL GET COUNT OF 0-3
0003 B60E7E             00940        LDA    XCOORD          GET X-COORDINATE IN A
0006 44                 00960        LSRA                   DIVIDE X COORD BY 2
0007 2401               00980        BCC    DONEX           IF B0 WASN'T SET (X EVEN),
                        01000 *                             DON'T INC B
0009 5C                 01020        INCB                   X ODD--SET B0 IN B
000A 3402               01040 DONEX  PSHS   A               SAVE X/2 TEMPORARILY
000C B60E7F             01060        LDA    YCOORD          GET Y-COORDINATE
000F 44                 01080        LSRA                   DIVIDE Y BY 2
0010 2402               01100        BCC    DONEY           IF Y EVEN, B HAS CORRECT
                        01120 *                             COUNT NOW
0012 CA02               01140        ORB    #2              SET B1 IN B SINCE Y ODD
0014 3402               01160 DONEY  PSHS   A               SAVE Y/2 TEMPORARILY
0016 8608               01180        LDA    #8              SET BIT IN 1 TO LSR
0018 5A                 01200 BITLP  DECB                   DEC CNT OF LSR'S
0019 2803               01220        BMI    GOTBIT          DONE WHEN B < 0
001B 44                 01240        LSRA                   SHIFT BIT IN A LEFT BY 1
001C 20FA               01260        BRA    BITLP           LOOP TILL A HAS 8,4,2, OR 1
001E B70E83             01280 GOTBIT STA    DOTBIT          STORE VALUE OF BIT TO SRP
0021 3502               01300        PULS   A               GET Y COORD / 2
0023 C620               01320        LDB    #32             PUT 32 IN B FOR MULT
0025 3D                 01340        MUL                    Y/2 * 32
0026 1F01               01360        TFR    X,D             X-FER RESULT TO X
0028 3502               01380        PULS   A               GET X COORD / 2
002A 3086               01400        LEAX   A,X             ADD X/2 TO Y/2*32
002C 30890400           01420        LEAX   1024,X          FORM ADDR TO GRAPHICS BLK
0030 A684               01440        LDA    0,X             GET WORD NOW IN THIS SPOT
0032 F60E81             01460        LDB    OPTION          GET OPTION (0-2)
0035 271E               01480        BEQ    SET             IF 0, OPTION=SET
0037 5A                 01500        DECB                   IS IT 1?
0038 2732               01520        BEQ    RESET           IF 1, OPTION=RESET
003A 5F                 01540        CLRB                   OPTION=POINT--SET RESULT
                        01560 *                             REG TO "OFF"
003B 4D                 01580        TSTA                   IS B7 OF WORD SET?
003C 2B04               01600        BMI    NOTALF          IF B7 SET, NOT ALPHA MODE
003E CAFF               01620        ORB    #$00FF          -1=ALPHANUMERIC MODE
0040 200E               01640        BRA    EXIPNT          STD RESULT & EXIT
0042 B40E83             01660 NOTALF ANDA   DOTBIT          IS THE BIT SET?
0045 2709               01680        BEQ    EXIPNT          IF RESET, RET 0 (IN B)
0047 E684               01700        LDB    0,X             GET WORD WE'RE TESTING
0049 C47F               01720        ANDB   #$007F          CLEAR BIT 7
004B 54                 01740        LSRB                   RT-JUSTIFY COLOR BITS 6-4
004C 54                 01760        LSRB                   SO COLOR POINT IS SET TO
004D 54                 01780        LSRB                   WILL BE RETURNED IN B
004E 54                 01800        LSRB
004F 5C                 01820        INCB                   B NOW HAS COLOR TO RET
0050 F70E82             01840 EXIPNT STB    RESULT          STORE RESULT OF POINT
0053 201F               01860        BRA    EXISRP          EXIT
0055 BA0E83             01880 SET    ORA    DOTBIT          SET BIT ON FOR DOT
0058 848F               01900        ANDA   #$008F          CLEAR COLOR BITS
005A F60E80             01920        LDB    COLOR           GET COLOR TO SET DOT
005D 5A                 01940        DECB                   DEC SO IN RANGE 0 - 7
005E 58                 01960        LSLB                   PUT COLOR IN BITS 6 - 4
005F 58                 01980        LSLB
0060 58                 02000        LSLB
0061 58                 02020        LSLB
0062 3404               02040        PSHS   B               PUT COLOR BITS ON STACK
0064 AAE0               02060        ORA    ,S+             OR INTO GRAPHIC WD; CLR STK
0066 8A80               02080        ORA    #$0080          SET GRAPHICS BIT ON
0068 A784               02100        STA    0,X             STD WORD IN VIDEO MEMORY
006A 2008               02120        BRA    EXISRP          EXIT
006C 730E83             02140 RESET  COM    DOTBIT          SET ALL BITS EXCEPT THE 1
006F B40E83             02160        ANDA   DOTBIT          RESET BIT FOR THE DOT
0072 A784               02180        STA    0,X             STORE WORD WITH DOT RESET
0074 3537               02200 EXISRP PULS   D,CC,X,Y        RESTORE SAVED REGS
0076 39                 02220        RTS                    RETURN
```

SAMPLE DATA STATEMENT OUTPUT FOR PONG/ASM:

```
100 DATA 410:REM==>SIZE OF OBJECT CODE
110 DATA 52,55,141,29,182,14,116,129
120 DATA 2,38,2,141,123,127,14,125
130 DATA 23,0,220,141,12,182,14,116
140 DATA 129,2,38,2,141,106,53,55
150 DATA 57,134,255,183,1,85,183,1
160 DATA 86,134,1,183,14,126,182,14
170 DATA 123,183,14,128,127,14,129,173
180 DATA 159,160,0,129,94,38,35,182
190 DATA 14,121,129,3,39,65,128,1
200 DATA 183,14,121,128,1,183,14,127
210 DATA 23,1,71,139,3,183,14,127
220 DATA 198,1,247,14,129,23,1,58
230 DATA 32,37,129,10,38,33,182,14
240 DATA 121,129,28,39,26,139,1,183
250 DATA 14,121,139,1,183,14,127,23
260 DATA 1,32,128,3,183,14,127,134
270 DATA 1,183,14,129,23,1,19,57
280 DATA 134,255,183,1,88,183,1,83
290 DATA 134,62,183,14,126,182,14,124
300 DATA 183,14,128,127,14,129,173,159
310 DATA 160,0,129,9,38,35,182,14
320 DATA 122,129,3,39,65,128,1,183
330 DATA 14,122,128,1,183,14,127,23
340 DATA 0,224,139,3,183,14,127,198
350 DATA 1,247,14,129,23,0,211,32
360 DATA 37,129,12,38,33,182,14,122
370 DATA 129,28,39,26,139,1,183,14
380 DATA 122,139,1,183,14,127,23,0
390 DATA 185,128,3,183,14,127,198,1
400 DATA 247,14,129,23,0,172,57,134
410 DATA 1,183,14,129,134,5,183,14
420 DATA 128,182,14,117,183,14,126,246
430 DATA 14,118,247,14,127,23,0,146
440 DATA 187,14,119,251,14,120,183,14
450 DATA 117,183,14,126,193,2,44,2
460 DATA 198,2,193,29,47,2,198,29
470 DATA 247,14,118,247,14,127,127,14
480 DATA 129,23,0,110,129,2,38,5
490 DATA 240,14,121,32,27,129,61,38
500 DATA 78,52,2,182,14,116,129,2
510 DATA 53,2,39,9,198,1,247,14
520 DATA 125,198,16,32,6,240,14,122
530 DATA 247,14,120,112,14,119,193,255
540 DATA 39,38,193,0,39,34,193,1
550 DATA 39,30,129,61,38,19,182,14
560 DATA 116,129,2,39,5,112,14,119
570 DATA 32,21,134,4,183,14,125,32
580 DATA 32,134,3,183,14,125,32,25
590 DATA 134,2,183,14,125,32,18,193
600 DATA 2,47,6,193,29,44,2,32
610 DATA 8,112,14,120,134,1,183,14
620 DATA 125,57
```

SAMPLE DATA STATEMENT OUTPUT FOR SETRES/ASM:

```
1000 DATA 119:REM==>SIZE OF OBJECT
CODE
1010 DATA 52,55,95,182,14,126,68,36
1020 DATA 1,92,52,2,182,14,127,68
1030 DATA 36,2,202,2,52,2,134,8
1040 DATA 90,43,3,68,32,250,183,14
1050 DATA 131,53,2,198,32,61,31,1
1060 DATA 53,2,48,134,48,137,4,0
1070 DATA 166,132,246,14,129,39,30,90
1080 DATA 39,50,95,77,43,4,202,255
1090 DATA 32,14,180,14,131,39,9,230
1100 DATA 132,196,127,84,84,84,84,92
1110 DATA 247,14,130,32,31,186,14,131
1120 DATA 132,143,246,14,128,90,88,88
1130 DATA 88,88,52,4,170,224,138,128
1140 DATA 167,132,32,8,115,14,131,180
1150 DATA 14,131,167,132,53,55,57
```

ZAP TO DISABLE LDOS 5.1 PASSWORD CHECKING - This was found lying around on a scrap of paper(!). I don't own a copy of LDOS, so I can't verify this, but you may try it if you like (let me know if it works!). To disable password checking in LDOS 5.1.x, change SYS2/SYS, Record 2, Byte 19H from 28H to 18H (surrounding bytes should be: 52 E1 xx 2D 79).

BOOK REVIEW - DR. DOBB'S JOURNAL, VOLUME SIX (Published by Hayden Book Company, Inc., $29.95) - Back in the days before anyone ever heard of the TRS-80 (1976, to be exact), a magazine called Dr. Dobb's Journal of Computer Calisthenics & Orthodontia began publication (their motto was, "Running Light Without Overbyte"). If I may risk the comparison, Dr. Dobb's was to the early microcomputer user what The Alternate Source Programmer's Journal was to the TRS-80 user - that is, a magazine for folks that got into such things as assembly language, operating systems, and the latest innovations in high-level languages. Each volume of Dr. Dobb's has been printed as a bound volume, and the latest in that series is bound volume six, which includes all of the issues of Dr. Dobb's that were published in 1981. The book itself is fairly hefty - with 558 pages of good, heavy paper stock, it's even thicker than an issue of BYTE.

To get right to the point, you may wonder if this volume contains much of interest to the TRS-80 user. That depends. If you're looking for programs that you can just type in and run, forget it - unless you happen to be a tape user and need a fast cassette I/O routine for BASIC arrays, because that is the only program specifically written for the TRS-80 in the entire book! There is also an article describing how to interface a TRS-80 Voice Synthesizer to other brands of microcomputers, and a few reviews of TRS-80 related products, but not much else.

However, if you're willing to try and adapt programs written for other machines, you can try your luck on several Z-80 programs (such as a file comparator, video drivers, a memory test, a floppy disk test, and a cross-assembler that will permit you to generate object code for an 8086 or 8088 CPU). There are also several short but useful Z-80 subroutines in various places around the book. Two cautions, however: some of the assembly listings use octal constants (rather than the more familiar decimal or hexadecimal), which may not be a problem for you, since many TRS-80 Editor-Assembler packages are capable of handling octal numbers provided that they are suffixed with the letter "O" (or whatever the assembler uses to indicate octal). The other problem is that some of the listings use 8080 style mnemonics, instead of the more familiar Z-80 opcodes. Most TRS-80 assemblers will choke on 8080 mnemonics! Have fun converting them to Z-80 format.

On the other hand, if your interest is in computer languages, you'll probably find that many of the articles in this book will catch your fancy, especially if your interest is in FORTH - there are a number of FORTH-related articles included in this volume. Other languages discussed include C, COMAL-80, Tiny BASIC, PARFOR, SLIC, PIDGIN, LISP, and various dialects of assembly languages.

Several program listings are included, most of which are utilities of one type or another, with a Rubik's Cube simulator in BASIC as a notable exception (and the only program in the book that even resembled a game). Many of the listings are demonstration routines, designed to show how to use a particular algorithm under discussion, or intended to illustrate a better way to program a specific task. This seems to be the general theme of Dr. Dobb's - it's definitely written for programmers and "true hackers" (I'm using the word "hacker" as computer people use it, not as Hollywood does).

My favorite part of this volume was a monthly column that began in the May, 1981 issue, entitled "Dr. Dobb's Clinic" and written by D. E. Cortesi. One purpose of this column was to solicit input from readers of the magazine, to provide solutions to various problems that were presented (for example, the fastest way to divide by 10 in assembly language, or how to define a MIN or MAX function in a version of BASIC that doesn't have those functions built in - such as TRS-80 BASIC). It was very interesting to see some of the clever answers that were provided to do the task faster, better, or with less memory usage.

Dr. Dobb's Journal began accepting advertising in 1981. The ad pages were removed from the bound volume, but in at least one instance, a couple of pages of program listings were (probably inadvertently) removed as well. Unfortunately, the missing pages were part of the "Dr. Dobb's Clinic" column in the November, 1981 issue (pages 60-61). Boo, hiss!

The bottom line - if you really get into the inner workings of microcomputers in general (not just the TRS-80), you'll love this book. If your interest is limited to the TRS-80, you may find it hard to justify spending $29.95 for the book, although it may be worth the money if you are really into languages such as FORTH, or if you'd enjoy the challenge of converting programs written for other Z-80 based computers to run on the TRS-80. If you aren't sure you want to spend the $29.95, you might want to buy or borrow an issue of Dr. Dobb's Journal and look it over, to get an idea of the type of material they publish. If you like the magazine, go ahead and buy the bound volume. If you like bound volume six, you may be interested to know that the previous five bound volumes are still available, at $25.95 each. I haven't seen any of the previous volumes, so I cannot comment on them. If you want further information about Dr. Dobb's Journal, you might try writing to the publisher: People's Computer Company, Box E, 1263 El Camino Real, Menlo Park, California 94025.

As an example of the type of interesting little routine I found in "Dr. Dobb's Clinic", here's something that originally appeared in the October, 1981 issue of Dr. Dobb's. It was sent by a reader in response to a challenge to find the fastest way to divide a number by ten in assembly language:

"Allen Ashley of Pasadena, CA sent in what is certainly the most unique division algorithm we've seen. It can handle only numbers less than 2560, for reasons that will become clear as you study the code. It's very quick, and could be useful as in-line code in certain special applications. Ashley's analysis runs thus: given that HL is equal to or greater than zero but less than 2560,

$$256*(HL/10) = 25*HL + (6*HL)/10$$
$$= 25*HL + HL/2 + HL/10$$

and his code to generate the latter expression is an education:"

```
; Ashley's Unique Division Routine
; INPUT: HL contains a positive binary integer less than 2560.
; OUTPUT: H only contains the input divided by 10.
; ALTERS: ALL
;
HLD10   LD      B,H
        LD      C,L
        CALL    HX25    ; HL = 25*HL
        OR      A
        LD      A,B
        RRA
        LD      B,A
        LD      A,C
        RRA
        LD      C,A
        ADD     HL,BC   ; HL = 25*HL + HL/2
        LD      E,H
        LD      D,0H    ; DE = (25.5*HL)/256, or about HL/10
        ADD     HL,DE   ; HL = 25*HL + HL/2 + HL/10
        INC     HL      ; (after fudge)
        RET             ; H = HL/10 when HL<2560
HX25    CALL    HX5
HX5     LD      D,H
        LD      E,L
        ADD     HL,HL
        ADD     HL,HL
        ADD     HL,DE
        RET
```

[Editor's note: I have changed the 8080 mnemonics used in the original listing to Z80 mnemonics in the above listing, since most TRS-80 assemblers accept Z80 mnemonics only. However, I did not optimize the code for use on the Z80. I suggest that Z80 users substitute SRL B and RR C instructions for the code that falls between (but not including) the CALL HX25 instruction and the ADD HL,BC instruction. This would further improve execution time and save a bit of memory, and would have the added benefit of saving the contents of the A register.]

8080/8085 TO Z80 CONVERSION CHART - The Z80 microprocessor is "upward compatible" with the older 8080 microprocessor - that is, programs written for the 8080 can run on the Z80, though the reverse may not be true. The 8085 has the same instruction set as the 8080, except that it has two added instructions (RIM and SIM - these read and set the 8085 "interrupt mask", so there is no Z80 equivalent of these two instructions). However, 8080/8085 assembly language uses different mnemonics than Z80 assembly, so conversion of published programs for the 8080/8085 becomes a difficult task unless you have a conversion chart. This is such a chart. The 8080/8085 mnemonic is found in the left column, the equivalent Z80 mnemonic on the right. Now you can convert those nice 8080/8085 subroutines that you sometimes find in non-TRS-80 magazines for use on your '80.

(NOTE: Register pairs are indicated by only the first letter of the register pair in 8080/8088 assembly language listings, thus INX H translates to INC HL.)

8080/8085 Z80		8080/8085 Z80		8080/8085 Z80	
ACI n	= ADC A,n	IN n	= IN A,(n)	RAL	= RLA
ADC r	= ADC A,r	INR r	= INC r	RAR	= RRA
ADC M	= ADC A,(HL)	INR M	= INC (HL)	RC	= RET C
ADD r	= ADD A,r	INX rr	= INC rr	RET	= RET
ADD M	= ADD A,(HL)	JC nn	= JP C,nn	RLC	= RLCA
ADI n	= ADD A,n	JM nn	= JP M,nn	RM	= RET M
ANA r	= AND r	JMP nn	= JP nn	RNC	= RET NC
ANA M	= AND (HL)	JNC nn	= JP NC,nn	RNZ	= RET NZ
ANI n	= AND n	JNZ nn	= JP NZ,nn	RP	= RET P
CALL nn	= CALL nn	JP nn	= JP P,nn	RPE	= RET PE
CC nn	= CALL C,nn	JPE nn	= JP PE,nn	RPO	= RET PO
CM nn	= CALL M,nn	JPO nn	= JP PO,nn	RRC	= RRCA
CMA	= CPL	JZ nn	= JP Z,nn	RST n	= RST n
CMC	= CCF	LDA nn	= LD A,(nn)	RZ	= RET Z
CMP r	= CP r	LDAX rr	= LD A,(rr)	SBB r	= SBC A,r
CMP M	= CP (HL)	LHLD nn	= LD HL,(nn)	SBB M	= SBC A,(HL)
CNC nn	= CALL NC,nn	LXI rr,nn	= LD rr,nn	SBI n	= SBC A,n
CNZ nn	= CALL NZ,nn	MOV r,r	= LD r,r	SHLD nn	= LD (nn),HL
CP nn	= CALL P,nn	MOV M,r	= LD (HL),r	SPHL	= LD SP,HL
CPE nn	= CALL PE,nn	MOV r,M	= LD r,(HL)	STA nn	= LD (nn),A
CPI n	= CP n	MVI r,n	= LD r,n	STAX rr	= LD (rr),A
CPO nn	= CALL PO,nn	MVI M,n	= LD (HL),n	STC	= SCF
CZ nn	= CALL Z,nn	NOP	= NOP	SUB r	= SUB r
DAA	= DAA	ORA r	= OR r	SUB M	= SUB (HL)
DAD rr	= ADD HL,rr	ORA M	= OR (HL)	SUI n	= SUB n
DCR r	= DEC r	ORI n	= OR n	XCHG	= EX DE,HL
DCR M	= DEC (HL)	OUT n	= OUT (n),A	XRA r	= XOR r
DCX rr	= DEC rr	PCHL	= JP (HL)	XRA M	= XOR (HL)
DI	= DI	POP rr	= POP rr	XRI n	= XOR n
EI	= EI	PUSH rr	= PUSH rr	XTHL	= EX (SP),HL
HLT	= HALT				

"USER FRIENDLY" BANNED BY UNICORN HUNTERS - The Unicorn Hunters of Lake Superior State College (located right here in good old Sault Ste. Marie, Michigan) have included the phrase "user friendly" among the words and phrases on their tenth annual dishonour list of words and phrases banished from the Queen's English for mis- or over-use, as well as general uselessness.

According to nominator Edward C. Loyer of the University of Michigan, "user friendly" comes "from the same folks who have given us 'up' meaning functioning and 'down' meaning broken."

The Unicorn Hunters further note that "a light switch which glows in the dark is 'user friendly.' A corkscrew is not. Mr. Loyer is concerned not with the 'user' but with the 'friendly!' 'Is this to give me the urge to take a particular system to lunch, or to find some alternative way to get intimate with it? Are there also systems or machines that are "user-unfriendly"?'"

Actually, the term "user friendly" wasn't the number one banishment for the year 1984. That honor goes to the term "high tech", which is said to be "used by politicians, advertisers, and educators to signify nothing except a vague jumble of concepts which they favor. Its most important contribution to the world of jargon is its potential for grammatical formulations. Does one use high tech like a wrench? Or operate it like a bulldozer? Practice it like a religion? Was high tech invented, developed, discovered or manufactured?'

For a copy of the 1984 banishment poster (price 50 cents) or further information about the Unicorn Hunters, phone (906) 635-2315 or write Unicorn Hunters, c/o Lake Superior State College, Sault Ste. Marie, Michigan 49783.

PERSONAL SOFTWARE MAGAZINE - The following item comes to us from the New York Amateur Computer Club newsletter:

...has anyone seen the new magazine, Personal Software? Describing itself as "The Monthly Review of the Best Packages", it takes the dubious position of not saying anything bad about anybody's product, ever. This is because, to paraphrase the editor, they've already checked it out and if there was anything wrong with it, at all, they wouldn't waste our time by reviewing it in the first place. A beneficial aside, of course, is that none of those companies with all those Big Ad Bucks ever get mad at Personal Software. Got it?

CHRISTIAN COMPUTER-BASED COMMUNICATIONS is a group you might want to be in contact with if your church owns or uses a microcomputer. They list their objectives as follows:

1) To spread the GOSPEL of God's Love among all people through means of the technology now available to us.

2) To aid new Christians with answers to questions of their Faith, Doctrines and Teachings.

3) To provide the secular world with Christian Alternatives to the Questions of Life.

4) To promote alternatives to the popular Video Games with Co-operative Games emphasizing Christian Moral Values.

5) To promote programs and teaching games which emphasize the Christ-Centered Life.

6) To aid Church administrators in their task as stewards of the Christian Body by providing computer-assisted aids to the efficient organization of their time.

7) To promote the use of Christian Computer Programs through distribution to associate members.

I received a rather lengthy letter from Mr. John Easton of Christian Computer/Based Communications, which indicates that their work actually goes quite a bit beyond what you might think by reading the above. I'd like to reprint one part of Mr. Easton's letter, which makes me suspect that at one time or another he's had contact with that unique breed known as "computer salespersons":

"... I suppose that brings me around to one of my minor peeves at Mainframe oriented thinking. Does one REALLY need to know how many times a parishioner's family has had Measles, and whether they individually and/or collectively enjoy the hobby of birdwatching? Quite apart from the ability to process such trivia (albeit perhaps not quite so tongue-in-cheek), the need of which many Mini systems salespersons appear to impress prospective clients with, it really quickly comes back to the person who is responsible for the upkeep of such records. After all, it is only as these multi-information databases are kept accurate, that we can use them for anything like the supposed whiz-bang manipulations we were originally sold on. I can't picture OUR parishioners bothering to inform the secretary of every minute change in 'church-roll status' (why they wouldn't probably be aware that much of the information existed on file, or that it was in any way necessary to the 'efficient' running of the church), and I certainly wouldn't like to be in the position of our local church secretary should the responsibility ultimately rest on HER shoulders to keep up the records!"

I reprinted the above because it is food for thought, for anyone maintaining a computerized database. Do we really NEED to file away every last bit of information about a person?

If you are interested in contacting Mr. Easton and/or obtaining further information about Christian Computer-Based Communications, you may write him at 44 Delma Drive, Toronto, Ontario M8W 4N6 or phone him at (416) 251-1511 (home) or (416) 965-1230 (work), or contact him through Compuserve # 71426,1371. I will mention that this organization does have some church-related software available for sale, but unfortunately, at the present time it is mostly written for Commodore machines (Commodore has a much better dealer support network in Canada than they do in the U.S.). However, the CCBC folks attempt to write these programs in 'plain vanilla' BASIC so that they can be easily transferred to other machines. They also apparently have a "once in a while" newsletter. If your church is using (or thinking about using) a computer, at least drop them a line and let them know what you're up to.

DISKETTE TIP - How many of you remember -T-C-S->, or the Tidewater TRS-80 Users Group (if you do, then you probably owned or used a TRS-80 Model I back before microcomputers became the "in" thing). Anyway, you may remember Les Logan, one of the driving forces behind that group. Well, Les is alive and well and running a disk drive alignment and repair business in Norfolk, Virginia, and he passes along this tip:

Do you know why alignment diskettes don't have hub rings? Because the rings and the diskettes they are "supposed" to protect have different temperature and humidity expansion coefficients (or maybe it is the adhesive that's different). Anyway, the rings don't return to the original shape after expanding/contracting with temperature and humidity changes as plain diskettes will, so they can cause randomly placed areas of head compliance and alignment problems, especially at 96 or 100 tracks per inch. The price you pay for durability is reliability, unless you maintain good temperature/humidity control.

MODEM PROTOCOL DOCUMENTATION by Ward Christensen - Many people ask me for documentation on my modem protocol [a popular error detecting and correcting protocol used by many Bulletin Board Systems -ed.], so here it is:

1/1/82 by Ward Christensen [last revision was 8/9/82]. I will maintain a master copy of this. Please pass on changes or suggestions via CBBS/Chicago at (312) 545-8086, CBBS/CPMUG (312) 849-1132 or by voice at (312) 849-6279.

NOTE this does not include things which I am not familiar with, such as the CRC option implemented by John Mahr.

At the request of Rick Mallinak on behalf of the guys at Standard Oil with IBM P.C.s, as well as several previous requests, I finally decided to put my modem protocol into writing. It had been previously formally published only in the AMRAD newsletter.

Table of Contents:

1. DEFINITIONS
2. TRANSMISSION MEDIUM LEVEL PROTOCOL
3. MESSAGE BLOCK LEVEL PROTOCOL
4. FILE LEVEL PROTOCOL
5. DATA FLOW EXAMPLE INCLUDING ERROR RECOVERY
6. PROGRAMMING TIPS.

1. DEFINITIONS: <soh> 01H, <eot> 04H, <ack> 06H, <nak> 15H, <can> 18H.

2. TRANSMISSION MEDIUM LEVEL PROTOCOL: Asynchronous, 8 data bits, no parity, one stop bit. The protocol imposes no restrictions on the contents of the data being transmitted. No control characters are looked for in the 128-byte data messages. Absolutely any kind of data may be sent - binary, ASCII, etc. The protocol has not formally been adopted to a 7-bit environment for the transmission of ASCII-only (or unpacked-hex) data , although it could be simply by having both ends agree to AND the protocol-dependent data with 7F hex before validating it. I specifically am referring to the checksum, and the block numbers and their ones-complement.

Those wishing to maintain compatibility of the CP/M file structure, i.e. to allow modemming ASCII files to or from CP/M systems should follow this data format:

* ASCII tabs used (09H); tabs set every 8.
* Lines terminated by CR/LF (0DH 0AH)
* End-of-file indicated by ^Z, 1AH. (one or more)
* Data is variable length, i.e. should be considered a continuous stream of data bytes, broken into 128-byte chunks purely for the purpose of transmission.
* A CP/M "peculiarity": If the data ends exactly on a 128-byte boundary, i.e. CR in 127, and LF in 128, a subsequent sector containing the ^Z EOF character(s) is optional, but is preferred. Some utilities or user programs still do not handle EOF without ^Zs.
* The last block sent is no different from others, i.e. there is no "short block".

3. MESSAGE BLOCK LEVEL PROTOCOL: Each block of the transfer looks like:

<SOH><blk #><255-blk #><--128 data bytes--><cksum>

in which:

<SOH> = 01 hex

<blk #> = binary number, starts at 01 increments by 1, and wraps 0FFH to 00H (not to 01)

<255-blk #> = blk # after going through 8080 "CMA" instruction, i.e. each bit complemented in the 8-bit block number. Formally, this is the "ones complement".

<cksum> = the sum of the data bytes only. Toss any carry.

4. FILE LEVEL PROTOCOL:

4A. COMMON TO BOTH SENDER AND RECEIVER: All errors are retried 10 times. For versions running with an operator (i.e. NOT with XMODEM), a message is typed after 10 errors asking the operator whether to "retry or quit".

Some versions of the protocol use <can>, ASCII ^X, to cancel transmission. This was never adopted as a standard, as having a single "abort" character makes the transmission susceptible to false termination due to an <ack> <nak> or <soh> being corrupted into a <can> and canceling transmission.

The protocol may be considered "receiver driven", that is, the sender need not automatically re-transmit, although it does in the current implementations.

4B. RECEIVE PROGRAM CONSIDERATIONS: The receiver has a 10-second timeout. It sends a <nak> every time it times out. The receiver's first timeout, which sends a <nak>, signals the transmitter to start. Optionally, the receiver could send a <nak> immediately, in case the sender was ready. This would save the initial 10 second timeout. However, the receiver MUST continue to timeout every 10 seconds in case the sender wasn't ready.

Once into a receiving a block, the receiver goes into a one-second timeout for each character and the checksum. If the receiver wishes to <nak> a block for any reason (invalid header, timeout receiving data), it must wait for the line to clear. See "programming tips" for ideas

Synchronizing: If a valid block number is received, it will be: 1) the expected one, in which case everything is fine; or 2) a repeat of the previously received block. This should be considered OK, and only indicates that the receivers <ack> got glitched, and the sender re-transmitted; 3) any other block number indicates a fatal loss of synchronization, such as the rare case of the sender getting a line-glitch that looked like an <ack>. Abort the transmission, sending a <can>.

4C. SENDING PROGRAM CONSIDERATIONS: While waiting for transmission to begin, the sender has only a single very long timeout, say one minute. In the current protocol, the sender has a 10 second timeout before retrying. I suggest NOT doing this, and letting the protocol be completely receiver-driven. This will be compatible with existing programs.

When the sender has no more data, it sends an <eot>, and awaits an <ack>, resending the <eot> if it doesn't get one. Again, the protocol could be receiver-driven, with the sender only having the high-level 1-minute timeout to abort.

5. DATA FLOW EXAMPLE INCLUDING ERROR RECOVERY: Here is a sample of the data flow, sending a 3-block message. It includes the two most common line hits - a garbaged block, and an <ack> reply getting garbaged. <xx> represents the checksum byte.

```
SENDER                                  RECEIVER
                                    times out after 10 seconds,
                          <---           <nak>
<soh> 01 FE -data- <xx>   --->
                          <---           <ack>
<soh> 02 FD -data- xx     --->     (data gets line hit)
                          <---           <nak>
<soh> 02 FD -data- xx     --->
                          <---           <ack>
<soh> 03 FC -data- xx     --->
   (ack gets garbaged)    <---           <ack>
<soh> 03 FC -data- xx     --->           <ack>
<eot>                     --->
                          <---           <ack>
```

6. PROGRAMMING TIPS:

* The character-receive subroutine should be called with a parameter specifying the number of seconds to wait. The receiver should first call it with a time of 10, then <nak> and try again, 10 times.

After receiving the <soh>, the receiver should call the character receive subroutine with a 1-second timeout, for the remainder of the message and the <cksum>. Since they are sent as a continuous stream, timing out of this implies a serious like glitch that caused, say, 127 characters to be seen instead of 128.

* When the receiver wishes to <nak>, it should call a "PURGE" subroutine, to wait for the line to clear. Recall the sender tosses any characters in its UART buffer immediately upon completing sending a block, to ensure no glitches were mis-interpreted.

The most common technique is for "PURGE" to call the character receive subroutine, specifying a 1-second timeout, and looping back to PURGE until a timeout occurs. The <nak> is then sent, ensuring the other end will see it.

* You may wish to add code recommended by John Mahr to your character receive routine - to set an error flag if the UART shows framing error, or overrun. This will help catch a few more glitches - the most common of which is a hit in the high bits of the byte in two consecutive bytes. The <cksum> comes out OK since counting in 1-byte produces the same result of adding 80H + 80H as with adding 00H + 00H.

MODEM FILE TRANSFER PROTOCOL by D.L. Covill is reprinted from Personal Systems (San Diego Computer Society Newsletter):

The file transfer protocol used in the MODEMx series of programs is a very good (although not perfect) protocol that has become something of a de facto standard, because it is in the

public domain and has been widely distributed on various remote CP/M systems in the country.

The protocol was originally developed by Ward Christensen, who wrote the original MODEM.ASM. It has since been enhanced by several others. This writeup is based on an article by Kelly Smith in CP/M News, January 1981, extended to include discussion of CRC and batch transfers.

WHAT IS A PROTOCOL AND WHY DO WE NEED ONE?

A protocol is, quite simply, the ritual courtesies exchanged by two separate but equal computer systems in order to insure an orderly and correct transfer of information. For short, interactive communications we can do without one because the information transferred is visible, and if it looks wrong or we lose it we can immediately ask for it again.

The situation is different when transferring data or program files. The volume to be transferred is greater, thus there is a greater chance of transmission errors. It's hard to tell whether an error has occurred (such as in COM files), and retransmitting the entire file would take too long and might only make things worse.

A protocol provides:

a. Positive control over the process and procedure.

b. A means of detecting errors in the transmission process.

c. The ability to retransmit any portion that has errors.

Note, however, that BOTH parties to the transmission must use the SAME protocol, or the ritual doesn't work. That's the reason for this document - so that people that want to (or have to) use a program other than MODEM7 can implement the XMODEM protocol and join the great army of public-domain transmitters.

CONTROL CHARACTERS USED

The following ASCII standard control characters are used in the protocol:

SOH	Start of Heading	01H	(CTRL-A)
ACK	Acknowledge	06H	(CTRL-F)
NAK	Negative acknowledge	15H	(CTRL-U)
EOT	End of Transmission	04H	(CTRL-D)
CAN	Cancel	18H	(CTRL-X)

GENERAL TRANSMISSION SCHEME

Data is sent in 128 byte numbered blocks, with a single checksum appended to each block. The receiving computer performs its own checksum as it acquires the incoming data, and upon completing each block compares its result with the checksum from the sending computer. If they match, it returns an ACK to the sender, meaning "received OK, send some more." If they don't match, a NAK (15H) is returned, meaning "that didn't look right, please send it again." This process continues until the entire file has been transmitted (or the number of errors causes one of the parties to give up).

Received data is stored in memory, then written to disk every 16 blocks (more or less - on the FORUM version, this is every 20 blocks).

BLOCK FORMAT

The sending computer transmits a block in the following form:

SOH Start of Heading	01H
Block #	8 bits
Complement of block #	8 bits
<128 data bytes>	8 bits each
Checksum	8 bits

The checksum is calculated by summing the SOH, block number, block number complement, and the 128 data bytes.

STARTING THE TRANSMISSION

The sending and receiving systems have to get "in sync" to start the transmission. This is easy - the sending computer simply waits for an initial NAK from the receiving computer. He's sure to get one, for the receiving computer will "time out" looking for data and send the NAK as a signal that he didn't receive a data block. The sending computer knows this and uses it as a signal to start the transmission.

Sending computer	Receiving computer
(waiting)	(waiting)
------	(times out)
------	NAK (15H)
Data block 1	--------
------	ACK (06H)
Data block 2	--------
etc	

RE-TRANSMITTING A BLOCK

What happens if the block is NAK'ed? Easy, the sending computer just re-sends the previous block.

Sending computer	Receiving computer
Data block 2 (errors)	NAK
Data block 2 (again)	ACK
Data block 3	etc

But what if the sending computer never receives the ACK (or NAK)? The sending computer times out after 10 seconds, decides that it has failed, and re-transmits the block. This is the reason for the block numbers - the receiver detects that this is the previous block all over again, throws it away, and returns an ACK, thereby catching up. The integrity of the block number is verified by summing the SOH (01H) with the block number plus the complement of the block number - this result must be zero for a proper transfer (e.g., for block 7, 01+07+F8=00).

The sequence of events then, looks like this:

Sending Computer	Receiving Computer
....	....
Data block 2	ACK (garbled)
Data block 2 (again)	ACK (but disregards duplicate data)
Data block 3	etc.

CONCLUDING A TRANSMISSION

Normal completion of a data transfer concludes with an EOT (End of Transmission, 04H) from the sending computer, with a final ACK from the receiving computer.

Sending Computer	Receiving Computer
Last data block	ACK
EOT	ACK
<END>	<END>

MODEM7 BATCH TRANSFERS

The MODEM7 program allows "batch" file transfers - the sending program says something like:

SB NEWSTUFF.*

and the receiving program says simply "RB."

The B sub-option means (a) that more than one file will be transmitted and (b) that the sending computer will send the file names as well as the data. What concerns us is the protocol for sending the file name.

(This function was added by someone other than Ward Christensen; it works, but isn't normal communications practice.) Basically, the sender sends the file name ONE CHARACTER AT A TIME, and the receiver ACK's each character separately. At the end, the sender sends an EOF (1AH) and the receiver replies with the checksum! If it matches, the sender sends an ACK, the receiver replies with a NAK, and we fall into the normal file transfer sequence.

Sending Computer	Receiving Computer
....	(awaiting file name)
	(times out) NAK
1st letter	ACK
2nd letter	ACK
....	....
11th letter	ACK
EOF (1AH)	checksum
ACK	NAK
Data block 1	etc

At the end of the data transmission, the receiver sends another NAK and gets either another file name or another EOT, indicating that there aren't any more. (Note: This may not be precisely correct in all details - the program is hard to read after all the volunteer labor on it!)

CRC CHECKING

MODEM7 also added a Cyclic Redundancy Check (CRC) option. This replaces the one-byte checksum with two bytes of CRC. CRC checking is much more reliable than checksums, giving better that 99.99% probability of error detection. If CRC checking is requested, the receiving system sends a 'C' in place of the initial NAK. If the sender responds within 3 seconds, the transfer

continues with CRC checking in effect. If there is no reply within 3 seconds, the receiver assumes the sender doesn't know about CRC checking (old version or different program), sends a NAK, and settles down to use the old checksum system. In batch mode, the decision about the first file carries forward for the others. MODEM7 and XMODEM are set up for CRC checking by default, but the user can specify either method at will.

ERROR ABORTS

The protocol will tolerate a reasonable amount of line noise, retransmitting when necessary. If the line quality is really trashy, however, after 10 retries (on the same block), the receiving computer will display "Retry or Quit?". If the operator enters "Q", a CAN (CANcel, 18H) is sent to cancel the entire transfer session.

PROBLEMS

Unfortunately, the protocol is not entirely bullet-proof. While the following cases are very rare, they can conceivably occur:

1. At end of transmission, if the receiving computer misses EOT, it will continue to wait for the next block (sending NAK's every 10 seconds, up to 10 times) and eventually "time out."

2. There is a possibility that an ACK could be "garbled" to a CAN, thus aborting prematurely.

Do not use the "V" (View) sub-option during file transfer with a slow (4800 baud or less) terminal or hard copy printer. It takes too long to write each character to these terminals, so you don't always get back in time to catch the next one from the modem. This should be readily apparent if it occurs.

USE FOUR DOUBLE-SIDED DISK DRIVES WITH A TRS-80 MODEL I? - I'm told that this is possible, but don't have the full details. However, I'll give you what I do have and perhaps some of you hardware hackers can take it from here.

The apparent problem is that the Western Digital FD1791 disk controller (or equivalent) IC used in most Model I double density adapters does not have side select output. Most DOSes "fudge" a side select by using the line that would normally control the fourth drive in a four-drive system for side select purposes (this probably isn't technically accurate but it's a close enough approximation for the moment). However, the FD1795 is an equivalent to the FD1791 with only one difference - pin 25 is defined as a side select output instead of a read gate. If a FD1793 controller was originally used (the equivalent of a FD1791 except has a true data bus instead of an inverted data bus), the FD1797 would be the replacement with side select.

Now comes the part that I'm really not sure of. I'm told that you can just pull the original controller IC, plug in the replacement with side select output, connect up a disk drive cable that has no pins pulled (that is, you do NOT want to use the stock Radio Shack disk drive cable), and you're ready to run double-sided with four drives. Well, maybe. But this leaves a whole host of unanswered questions, such as what are the requirements for the disk operating system?

So, here's a challlenge to you hardware hackers with Model I TRS-80s. How about setting up your system to run four double-sided drives, using as much of the above info as you can get to work for you. Keep track of the steps you had to go through to make it work (including how you configured your DOS), and send it to me (also please mention what brand of double density adapter you're using, if you know). I'll print the info here in Northern Bytes. Maybe some of us can start using our old Model Is at full disk drive capacity!

MISCELLANEOUS MODEL 4 INFORMATION from various sources (authors unknown):

The following patches can be applied to your TRSDOS 6.0 working disks to make life a little easier. They should not be applied to your master disk.

```
;Patch for TRSDOS  6.0
;To force file allocation to start from track 1 upward
;instead of using the random allocation it comes with
;Apply the patch SYS0/SYS.LSIDOS
 D04,E1=2E 01 00 00 00 00:F04,E1=D5 CD B8 06 D1 6C
;End of Patch

;Patch to prevent the DIR cmd from clearing the screen
;Apply to SYS6/SYS.LSIDOS
 D07,55=0D:F07,55=1C
;End of Patch

;The following will replace the Lib Command Word "REMOVE"
;with the word "KILL" to maintain consistancy in the method
;of getting unwanted files off your disks.
;Apply to SYS1/SYS.LSIDOS
 D01,C8=4B 49 4C 4C 20 20:F01,C8=52 45 4D 4F 56 45
;End of Patch
```

The following patch is by Guy Omer:

```
;Patch to Re-format 'LOG-ON' and 'MENU' of COMM/CMD 6.0.0
;(C) 1983 Guy Omer
;UPDATE: 06/12/83
D07,1B=4D 45 4E 55
F07,1B=20 3F 3F 3F
D09,FD=3E 3C 38
F09,FD=2D 38 3E
D0A,00=3E 20 66 6F 72 20 4D 45 4E 55
F0A,00=20 66 6F 72 20 6D 65 6E 75 0A
```

The following patch is by Jimmy Nord:

```
; FSTBOOT/FIX - 07/28/83
; This patch modifies the BOOT/SYS module to
; increase the speed of read/write disk i/o.
; To execute patch type:
; 'PATCH BOOT/SYS.V6KCZA USING FSTBOOT/FIX'.
; To remove patch type:
; 'PATCH BOOT/SYS.V6KCZA USING FSTBOOT/FIX (REMOVE)'.
; Note: This patch will make the clock less accurate
;       and type-ahead may lose characters during disk I/O.
; Patch by Jimmy Nord  [70605,373]
;
D0C,A5=0E F3 CD 6B 0F 1E 16 DB F0 A3 28 FB ED A2 7A
F0C,A5=0E CD 6B 0F 1E 16 DB F0 A3 28 FB ED A2 F3 7A
D0D,56=00 00 00
F0D,56=CD 00 05
D0D,F5=FA F3 CD 6B 0F 1E 76 DB F0 A3 28 FB ED A3 DB
F0D,F5=FA CD 6B 0F 1E 76 DB F0 A3 28 FB ED A3 F3 DB
; end of speed patch.
```

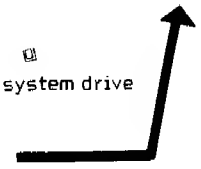
The passwords for TRSDOS 6.0 are as follows:

/SYS - LSIDOS	or	DB0
/FLT - GHNA	or	WHM2
/DRV - V71A	or	UBJ
/CMD - W0CK	or	QJ60
/BASIC - BASIC	(includes overlays)	
/DCT - W0CK	or	QJ60

The first of these are the "Real" passwords, the second, decoded with SU+

The following /JCL file can be used to move TRSDOS 6.x /SYS programs into the MEMDISK:

```
;
;
;
;
//alert 3,3,3,3
; Creating MEMDISK in Banks 1 and 2
;
system (drive=2,driver="memdisk")
d
d
y
;
;
;
//alert 6,6,6,6,0,0,0,0
;
;Moving system files to MEMDISK
backup $/sys:0 :2 (sys)
;
;
;
;
;Press <ENTER> to swap system drive
//alert (3,3)
;
system (system=2)
;
;
//alert 4,4
; Write protecting MEMDISK
;
system (drive=0,wp)
;
//exit
```



If you want to format double sided disks, just specify the following in the format command line:

FORMAT :? (SIDES=2)

with the ? being the drive you want to use.

The Model 4 BASIC does have a sound command, even if it's not in the book. The syntax is SOUND (X,Y) where X is the tone (1 thru 7) and Y is the duration. It will accept 1/2 second intervals in the duration.

Here are a pair of BASIC speed change subroutines for use in the Model III mode. You've probably seen the wrong way to do this in other publications, well, here's the correct way:

```
5 ' THESE TWO LINES USED AS SUBROUTINES WILL TURN ON
THE 4 MHZ CLOCK ON A MODEL 4 (LINE 8000) AND TURN IT
BACK OFF (LINE 9000)
6 ' THESE MAY BE USED AS SUBROUTINES WITH APPROPRIATE
GOSUBS BEFORE AND AFTER DISK I/O.  NEWDOS/80 HAS
PROVISIONS FOR HIGHER SPEED CLOCKS (SYSTEM OPTION BJ).
TURN HER ON AND LET HER RUN.
8000 X=PEEK(16912): X=X OR 64: POKE 16912,X: OUT 236,X:
RETURN
9000 X=PEEK(16912): X=X AND 191: POKE 16912,X: OUT 236,X:
RETURN
```

PATCHES TO ALLOW SOME PROGRAMS TO WORK WITH THE MODEL III VERSION OF NEWDOS/80 by Bob Dratch:

Special patches to TRS-80 MOD III transferred programs so that they will work with NEWDOS/80 version 2.0: TRSDOS does not use the same <BREAK> key initialization as does NEWDOS. Therefore, certain programs that you transfer over from TRSDOS to NEWDOS will not work properly. The following zaps will allow <BREAK> key operation. Use SUPERZAP in the DFS mode to apply the following zaps:

```
1) PROGRAM:  FORTRAN/CMD  (also known as EDIT/CMD)
Filespec: EDIT/CMD
Sector 02, byte DAH from AF 32 1E 53
                    to C3 A8 7C 00
Sector 43, byte 54H from 00 00 00 00 00 00 00 00 00 00 00 00
                    to AF 32 1E 53 3E C9 32 78 44 C3 02 54

2) PROGRAM:  F80/CMD  (within the FORTRAN package,
Compiler)
Filespec: F80/CMD
Sector 00, byte 04H from C3 A1 57 00 00 00 00 00
                    to 3E C9 32 78 44 C3 A1 57

3) PROGRAM  L80/CMD  (within the FORTRAN package, Linker)
Filespec: L80/CMD
Sector 00, byte 07H from AF 32 15 43
                    to C3 E6 74 00
Sector 35, byte 72H from 30 00 00 00 00 00 00 00 00 00 00 00
                    to 3E C9 32 78 44 AF 32 15 43 C3 07 52

4) PROGRAM  Series I Editor Assembler - Disk (This is a
prettier program than is supplied on your NEWDOS/80 operating
disk, and is supplied by Radio Shack
Filespec: EDTASM/CMD
Sector 02, byte 00H from 2A 0D 40
                    to C3 EA 5B
Sector 05, byte 80H from 4C 69 63 65 6E 73 65 64 20 74 6E 20
                    to AE 3E C9 32 78 44 2A 0D 40 C3 6B 58
```

The purpose of these zaps is to reload the BREAK enable function of NEWDOS/80, which TRSDOS generally fails to do.

3E,C9,32,78,44 is the group of HEX bytes to be patched into appropriate areas of the various TRSDOS, and other programs that require the <BREAK> key to work. You should be able to patch other TRSDOS programs. By using LMOFFSET/CMD to locate the program's entry point and looking at the program, a careful choice of patch can be determined. I hope that this helps those that want to use the great features of NEWDOS/80 with their TRSDOS programs.

Bob Dratch 06/12/82

NEWDOS/80 PDRIVE SETTINGS - The following information was placed on the Lansing, Michigan CompuNet BBS by SYSOP Gordon Williams. The PDRIVE settings shown below can also be used on the Model I, provided that when TD=E, the letter A in specification TI=A... must be changed to C, D, or E depending on whether a Percom, Radio Shack, or LNW type double-density modification is used:

PDRIVE settings for NEWDOS80 version 2 Model 3 -- to be used when you want to read/write diskettes written under other DOS's (compiled from information provided by Greg Small and Dennis Hill, two fellow Sysops... Thanks, fellas!)

```
MODEL III DISKS:
To read/write Model III TRSDOS v 1.3 (DD):
   TI=AM,TD=E,TC=40,SPT=18,TSR=3,GPL=6,DDSL=17,DDGA=2
To read/write Model III DOSPLUS and LDOS (DD):
    TI=A,TD=E,TC=40,SPT=10,TSR=3,GPL=2,DDSL=20,DDGA=2

MODEL I DISKS:
To read/write Model I TRSDOS v 2.3, DOSPLUS, LDOS (SD):
    TI=A,TD=A,TC=35,SPT=10,TSR=3,GPL=2,DDSL=17,DDGA=2
To read/write Model I NEWDOS80 v 2 (DD):
   TI=AK,TD=E,TC=35,SPT=18,TSR=3,GPL=2,DDSL=17,DDGA=2
To read/write Model I DOSPLUS and LDOS (DD):
    TI=A,TD=E,TC=35,SPT=10,TSR=3,GPL=2,DDSL=17,DDGA=2
```

NOTE: If Model I disks were written with 40 tracks, change "TC" to 40 [and, for DOSPLUS or LDOS, change DDSL to 20 -ed.]. "DD" and "SD" are abbreviations for Double Density and Single Density respectively. I have not tested these settings on an LDOS disk, but they should work since DOSPLUS and LDOS can read each other.

To change the PDRIVE table on the disk in drive 0, and to make the changes effective for diskettes placed in drive 1:

PDRIVE 0 1 TI=C TD=E ..... DDGA=2 A

If anyone knows how to set PDRIVE for 80 track DOSPLUS and LDOS diskettes, please contact Gordon Williams at CompuNet (517-339-3367) or Dennis Hill at Babblenet (517-485-6232).

Gordon Williams Oct. 10, 1983

[Editor's note: According to a recent article by Alan Abrahamson in the Voice of the '80 Newsletter, a Model I MULTIDOS "P" Density disk would take the same PDRIVE settings as a standard Model I NEWDOS/80 (DD) disk, except that DDSL would be set to one-half of the TC value]

MAX-80 PATCH FOR TASMON - The following LDOS patch file allows TASMON to run on a Lobo MAX-80:

```
,LTASMON/FIX - 02/12/83
,Patch to TASMON/CMD to allow use of LDOS *KI *DO and *PR
,drivers allowing TASMON to run on Lobo MAX-80
X'7933'=D5 11 15 40 FB CD 13 00 F3 D1 C9
X'7A1C'=D5 11 1D 40 CD 1B 00 D1 C9
X'7A45'=D5 11 25 40 CD 1B 00 D1 C9
,End of patch
```

ARE YOU BORED? Here's an item I can agree with (at least sometimes) - it's excerpted from "I/O News Update" by Si Hawk in The I/O Port:

Programming is a boring job! That was the finding by a group of researchers at the University of Colorado. After examining data from more than 6,000 computer programmers the group found that over 50% found their jobs boring and unchallenging. The researchers further declared that by making programming jobs more satisfying the resulting increase in programmer productivity would amount to 10% to 40%.

BOOBY TRAPS is reprinted from the Winnipeg Micro-80 User's Group newsletter:

I was given an article about Software Piracy that mentioned a new way I hadn't heard of. Here's a bit of info from the article:

BOOBY TRAPS - a particularly insidious variety of psychological warfare that either lets you spend hours creating files on your pirated program, before NEWing your diskette, or (even worse) a new breed of diskette with holes punched strategically around the perimeter to ENSURE that it will be the LAST diskette you copy on that drive.

The last one is used on some (expensive) business software packages. The label on the disk clearly states:

THIS DISK IS COPY PROTECTED. Any attempt to duplicate may cause permanent damage to disk drive.

It appears that this is no idle threat. By rotating the diskette in its sleeve, one finds a 3 mm hole punched in the magnetic media, near the outside edge. If the diskette was duplicated, the drive would attempt to read the whole diskette and catch the hole, which could tear the diskette and/or bounce the recording head mechanism off the platter, PERMANENTLY DAMAGING the disk drive.

I've seen the first one operate but haven't run into the second. Forewarned is forearmed.

COMPUTER OPPORTUNITIES

How to make it on your own with computers. New product forecasts, used equipment and residual value studies, price/performance studies, DP spending analyses, evaluation of acquisition methods. What's new and what's best in hardware, software, courseware, databases, networking, publications, grants, videodisks and consortia. Money-making opportunities for microcomputer owners. Business ideas, opportunities, pitfalls to beware, marketing hints and sources for the micro-entrepreneur. The booming computer industry has produced an explosion in hardware and software that offers confusion to some computer owners and enlightening predictability to others. The key to business success in the computer business is good information. Yet the business experience from which good information is derived can be very expensive. The experts who publish the services offered in this unique package are professionals with long experience and who devote their full time to their work. Here's how you can obtain the fruits of their experience for less than 90¢ apiece. Through this bargain-packed offer, SELECT INFORMATION EXCHANGE, America's leading financial subscription agency, makes available to you a selection of 16 sample subscriptions to a variety of money-making, money-saving DP publications, magazines and newsletters. Their regular subscription prices range as high as $295 apiece. Your price for the entire package under this amazing one-time-only offer is $14, or about 90¢ apiece. Exact descriptions of each of the 16 publications are below. To order the entire package of 16 sample subscriptions, merely complete and return the coupon below, along with your $14 remittance.

1. FINANCIAL & INVESTMENT SOFTWARE REVIEW. Receive a sampling of articles from past issues. Contains investment software reviews, and unique articles on investing strategy, with a special focus on micro-computerized investing. **1 Yr, $60.**

2. SMALL BUSINESS COMPUTERS. The practical computer magazine for businespeople. Offers clear solutions to problems common to microcomputer users. Includes in-depth reviews of equipment and software, industry news, and new products. A non-technical approach for individuals considering a computer purchase as well as for those who already own a system. **1 Yr, $12.97.**

3. COMPUTER ECONOMICS REPORT. The financial advisor of DP users: new products forecasts, current third party lease prices, used equipment prices and residual values forecasts, price/performance charts, tax and law factors, data processing spending analyses, evaluation of acquisition methods. **1 Yr, $295.**

4. SCOPE. Designed for developers and users of software for instruction, research, and communication on the university level. Reports on hardware, courseware, databases, networking, publications, campus news, grants, videodisks and consortia. Especially useful is an international calendar of related conferences. **1 Yr, $47.**

5. COMPUTER EXECUTIVE LETTER. Information to assist and support the decisions of DP executives: cost of ownership, analyses of new product announcements, DP personnel salaries, evaluation of price versus performance, forecast of future products, residual value forecasts, and more. **1 Yr, $195.**

6. COTTAGE COMPUTING. Monthly magazine designed to help microcomputer owners cash in on the booming computer industry. Each issue is packed with factual articles, news, trends, case studies, money-making opportunities, pitfalls to beware, and much more. Learn how others are making cash with their computer and how you can join them. **1 Yr, $14.**

7. COMPUTER FRAUD & SECURITY BULLETIN. Reports on techniques of computer crime, security management, management audit and financial control, recruitment and employee screening, new security hardware and software. Written for auditors, inspectors, DP managers, financial and corporate management in computer using companies —especially banks, finance houses, etc. **1 Yr, $190.**

8. COMPUTER SHOPPER. A buy, sell and trade publication for computer equipment and software. Bargains from individuals and dealers worldwide. Approximately, 84 big 11" × 14" pages every month. **1 Yr, $10.**

9. MICRO MOONLIGHTER NEWSLETTER. Devoted exclusively to aiding the owner of a personal computer in the creation, building, and maintenance of a home-based business. Contains business ideas, marketing hints, and sources for the micro-entrepreneur. **1 Yr, $25.**

10. MICRO M.D. JOURNAL. Focuses on Hi-Tech and Medical stocks with comprehensive analysis of stability and growth potential using highly successful measures of market momentum. Provides reviews of new financial software for tax management, stock investment and commodities trading. In depth reports on computer applications for the health professional. **1 Yr, $60.**

11. COMPUTER DESIGN: Serves those companies or organizations concerned with design and application of Digital Equipment and Systems in computing, data processing, control and communication. This field covers engineering activities being applied in industry, government, the Military, business and the sciences. **1 Yr, $50.**

12. EDP WEEKLY. Industry Reports Inc. The top news in the data processing industry, plus digest of new contract awards. Government procurements, of orders, installations and applications of EDP equipment; highlights of Wash ington events that affect EDP business; listings of new products and literature; business, financial and corporate news, major personnel changes and upcoming events. **1 Yr, $120.**

13, PERIPHERALS DIGEST. Industry Reports Inc. A sophisticated newsletter directed to Executives in Electronic Data Processing covering converters, data communicators, disc, displays, drums, OCR, plotters, printers, remote computing, tape and time-sharing. **1 Yr, $88.**

14. SOFTWARE DIGEST. Industry Reports Inc. A bi-weekly roundup of significant developments in all areas of the computer software industry. **1 Yr, $88.**

15. WORLD SOFTWARE MARKETS. Coverage of microcomputer software markets world-wide. Overseas business opportunities for software publishers distributors. Market trends. Turnkey projects. Overseas publishers distributors seeking franchises, licensing, joint-venture development. School use of microcomputers. Leading machines. Distribution and marketing channels. **1 Yr, $60.**

16. BUSS. Covers Heath Kit and Zenith micro-computers with announcements of new Heath Zenith compatible products, reports of user's experiences with their systems, news about Heath Zenith corporate activities and user community events. "Assistance Wanted" and "For Sale" columns. **1 Yr, $28**

FREE

FREE COMPUTER MAGAZINES? Answer this questionnaire!
Own personal computer? (A) Yes (B) No Plan to buy within year? (C) Yes (D) No Want free PC magazines? (E) Yes (F) No. Any children under 18? (G) Yes (H) No Brand owned or to be bought. (I) Adam (J) Apple (K) Atari (L) Commodore (M) Compaq (N) DEC (O) Epson (P) IBM (Q) Kaypro (R) NEC (S) Osborne (T) Pet (U) Radio Shack (V) TI (W) VIC (X) Vector (Y) Xerox (Z) Zenith (2) Other

The Alternate Source Package #48
704 North Pennsylvania Avenue
Lansing, Michigan 48906

YES. Please send me your package of sample subscriptions to 16 different specialized computer advisory services as described above. Although the regular annual subscriptions range as high as $295 apiece, I am enclosing only $14 or about 90c apiece, as complete payment for the package.

NAME _____

ADDRESS _____

CITY _____ **STATE** _____ **ZIP** _____

NOTE: Publications listed in this ad are those participating in the offer at the time the ad was prepared. Over the course of time, many of those listed might discontinue or withdraw from the offer. When this occurs, other publications of the same general substance are substituted. Therefore, some of the publications actually received under the offer might be different from some of those listed in the ad

MODULA-2!

NOW AVAILABLE

The Alternate Source Information Outlet is now the North American distributor for the Hochstrasser Computing Modula-2 System for Z80 CP/M! If you thought that Modula-2 wasn't possible on an 8-bit machine, keep reading! The programmer goals were to create a full Modula-2 compiler for an 8-bit machine that generates a reasonably small amount of code in a reasonably small amount of time leading to reasonably short execution times. These goals are now realities!

WHAT MAKES MODULA-2 SO GREAT?

In several PASCAL dialects, there exists possibilities to split programs apart into different "modules". These modules are compiled as if they were complete programs just lacking the main program. You can freely use procedures declared in one such "module" in others. All you have to do is to declare how this procedure looks and indicate that it is "external". It is a tool that is indispensable for the successful mastery of large programming projects. It is especially helpful for projects that are carried out by groups of programmers. In programmer terms, Modula-2 takes the concept of "local" and "global" variables to new dimensions. The programmer has complete control.

THE DOCUMENTATION

Manual Release 3-28-85/pwh -- almost 300 pages -- is very complete and well indexed (nine pages of carefully thought out subjects and subdivisions). It is presumed that the user has a some familiarity with PASCAL. Please note that the current documentation has been updated since an older version of this product was critiqued in an issue of BYTE. For persons who get squeemish at the thought of spending money, we have constructed a "Modula Sales Kit" which includes the complete Table of Contents and other information from the manual. One of these is free upon request.

POWER IS WHAT YOU GET

The Modula-2 Compiler Package for Z80 CP/M includes a linker, a reference lister, a converter, a system configuration package, two libraries (detailed below) and three test programs. Complete step-by-step instructions are included for assembling the test programs. The System generates fast, ROMable, reentrant Z80 native code. Assembly language integration is supported, as well as assembly language compiler output.

SOFTWARE SUPPORT

Included with this version of Modula-2 is a complete set of source code library routines including TERMINAL, SEQIO, TEXTS, REALTEXTS, INOUT, REALINOUT, MATHLIB, SYSTEM, ASCII, CHAINING, CMDLIN, CONTROLS, STRINGS, LONGSETS, CONVERSIONS, CONVERTREAL, FILENAMES, FILESYS, FILES, MOVES and OPSYS. The new documentation includes a START-UP GUIDE, INTRODUCTION TO MODULA-2, IMPLEMENTATION GUIDE, ADVANCED PROGRAMMING GUIDE and APPENDICES that detail error messages, object code format considerations, reserved words and symbols, the ASCII character set, language definition, a bibliography and 9 pages of index. A special section is devoted to Programming With Better Efficiency. The documentation is filled with small sample Modula-2 programs.

SYSTEM CONSIDERATIONS

Please note that this is a large system. The compiler itself uses about 170k of disk space. It is desirable to have at least two drives holding 350 Kilobytes of disk storage each to work comfortably with the system. A single double-sided eighty (holding 700k) would be ideal. Unless otherwise requested, the Modula-2 Compiler for Z80 CP/M will be shipped using the Montezuma Micro 40-track, single-side format. The compiler is only tested under CP/M 2.2. Eight inch CP/M formats are also available upon request.

FOR NEW MODULA-2 PROGRAMMERS

Folks near the mid-Michigan area: This summer we are planning at least one seminar detailing the use of Modula-2. We have a special package price on the seminar, which includes the price of the compiler and complete documentation, along with PERSONAL INSTRUCTION and enough diskettes to make a backup of your software and for working/scratch purposes. The price of the Modula-2 Compiler Package for CP/M, including the seminar and all mentioned above is $200. This price requires that you bring YOUR OWN CP/M computer. The price for the seminar with OUR computer is $300. The price for the software without the seminar is $165. The seminar without the software (you provide your own Modula-2 compiler and computer) is $59.95 if you make your reservations by August 1st, $99.95 after that date. The 8-hour seminar will take place in Lansing on Saturday, August 10th, and include lunch. Advanced seminars are planned and will depend on the success of this seminar, naturally.

WHERE TO GET IT!

THE ALTERNATE SOURCE

TAS I/O
704 N. Pennsylvania
Lansing, MI 48906
(517) 482-8270

NORTHERN BYTES

Subscription Information

Northern Bytes is edited by Jack Decker and published on an irregular basis by The Alternate Source Information Outlet. Back issues are available starting with Volume 5, Number 1. Issues prior to that are not available. Some of the most valuable articles from earlier issues may be reprinted in future issues of Northern Bytes. Currently there are eight back issues available for Volume 5, as well as all issues from Volume 6. All back issues are $2 each.

It is very easy to be placed on the Northern Bytes REGULAR list. Simply place your address, Visa or MasterCard number and expiration date on file with us. We will start with the issue you request. We do not bill you for ANY ISSUE until that issue has been mailed. This way, we can continue to offer you top quality information with absolutely no risk to you. There's no question of what to do about unfulfilled issues if we decide to quit publishing. Unless otherwise requested, we presume your subscription will extend through the month of your expiration date.

Don't have a charge card, huh? We understand the myriad of reasons for not having them and we feel that a "To-Be-Invoiced" policy could help increase the demand for Northern Bytes. If you'll do it for us, we'll do it for you. Would you like to be placed on a regular list TO BE BILLED for each issue? You could then send a check for the issues as they are mailed. If you didn't send a check, we would presume that your interest has died and discontinue your subscription. The only requirement for getting onto the list is to pay for the first issue up front; the next will be mailed automatically, if you request. You are insured that you will receive top of the line TRS-80 information as it is released. Ask to be placed on the NO RISK "To-Be-Invoiced Northern Bytes List".

Call or write, but SIGN UP TODAY!

The Alternate Source Information Outlet
704 North Pennsylvania Avenue
Lansing, MI 48906
(517) 482-8270

NORTHERN BYTES
c/o Jack Decker
1804 West 18th Street
Lot # 155
Sault Ste. Marie, Michigan 49783
MCI Mail Address: 102-7413
Telex: 6501027413
(Answerback: 6501027413 MCI)

Bulk Mail
U.S. Postage Paid
Permit #815
Lansing, MI

POSTMASTER: If undeliverable return to:
The Alternate Source, 704 North Pennsylvania Avenue, Lansing, Michigan 48906

To: